SOFTWARE REFERENCE MANUAL

PMAC / PMAC2

Programmable Multi-Axis Controller

3Ax-602204-xSxx

September 12, 2006

DELTA TAU
Data Systems, Inc.
NEW IDEAS IN MOTION ...

Single Source Machine Control *Power // Flexibility // Ease of Use*
21314 Lassen Street Chatsworth, CA 91311 // Tel. (818) 998-2095 Fax. (818) 998-7807 // www.deltatau.com

Copyright Information

To report errors or inconsistencies, call or email:

Delta Tau Data Systems, Inc. Technical Support
Phone: (818) 717-5656
Fax: (818) 998-7807
Email: support@deltatau.com
Website: http://www.deltatau.com

Operating Conditions

All Delta Tau Data Systems, Inc. motion controller products, accessories, and amplifiers contain static sensitive components that can be damaged by incorrect handling. When installing or handling Delta Tau Data Systems, Inc. products, avoid contact with highly insulated materials. Only qualified personnel should be allowed to handle this equipment.

In the case of industrial applications, we expect our products to be protected from hazardous or conductive materials and/or environments that could cause harm to the controller by damaging components or causing electrical shorts. When our products are used in an industrial environment, install them into an industrial electrical cabinet or industrial PC to protect them from excessive or corrosive moisture, abnormal ambient temperatures, and conductive materials. If Delta Tau Data Systems, Inc. products are exposed to hazardous or conductive materials and/or environments, we cannot guarantee their operation.

REVISION HISTORY				
REV.	**DESCRIPTION**	**DATE**	**CHG**	**APPVD**
1	INCORPORATED V1.17C FIRMWARE ADDENDUM	09/12/06	CP	C. WILSON

Table of Contents

PMAC COMMAND AND VARIABLE SUMMARY

Notes

PMAC syntax is not case sensitive.
Spaces are not important in PMAC syntax, except where noted
{} – item in **{}** can be replaced by anything fitting definition
[] – item in **[]** is optional to syntax
[{item}...] – indicates previous item may be repeated in syntax
[..{item}] – the periods are to be included in the syntax to specify a range
() – parentheses are to be included in syntax as they appear

Definitions

constant – numerically specified non-changing value
variable – entity that holds a changeable value
I-variable – variable of fixed meaning for card setup and personality (1 of 1024)
P-variable – global variable for programming use (1 of 1024)
Q-variable – local var. (in coord. sys.) for programming use (1 of 1024)
M-variable – variable assigned to memory location for user use (1 of 1024)
pre-defined variable – mnemonic that has fixed meaning in card
function – SIN,COS,TAN,ASIN,ACOS,ATAN,ATAN2,LN,EXP,SQRT,ABS,INT
operator – for arithmetic or bit-by-bit logical combination of two values:
+, -, *, /, % (mod), & (and), | (or), ^ (xor)
expression – grouping of constants, variables, functions, and operators
data – constant w/out parentheses, or expression w/ parentheses
comparator – evaluates relationship between two values: =, !=, >, !>, <, !<, ~, !~
condition – evaluates as true or false based on comparator(s)
simple condition – {expression} {comparator} {expression}
compound condition – logical combination of simple conditions
motor – element of control for hardware setup; specified by number
coordinate system – collections of motors working synchronously
axis – element of a coordinate system; specified by letter chosen from X, Y, Z, A, B, C, U, V, W
buffer – space in user memory for program or list; contains up to 256 motion programs and 32 PLC blocks

On-Line Commands

(Executed immediately upon receipt by PMAC)

On-line Global Commands

Addressing Mode Commands

@n – Address card n (n is hex digit 0 to f); serial host only
@ – Report currently addressed card to host; serial host only
#n – Make Motor n currently addressed motor
– Report currently addressed motor number to host
&n – Make coord. sys. n the currently addressed coord. sys.
& – Report currently addressed coordinate system to host

Communications Control Characters

<CTRL-H> – Erase last character from host (backspace)
<CTRL-I> – Repeat last command from host (tab)
<CTRL-M> – End of command line (carriage return)

<CTRL-N> – Report checksum of current command line
<CTRL-T> – Toggle serial communications full/half duplex
<CTRL-W> – Execute ASCII command from DPRAM buffer
<CTRL-X> – Abort current PMAC command and response strings
<CTRL-Y> – Report last command to host; ready to repeat to card
<CTRL-Z> – Make serial port the communications port

General Global Commands

$$$ – Global reset: including all motors and coord. systems
$$$*** – Reset and re-initialize entire card
PASSWORD={string} – Set/confirm password for PROG1000-32767,PLC0-15
SAVE – Save I-variables into non-volatile memory
UNDEFINE ALL – Erase definition of all coordinate systems
CLEARFAULT – Clear Geo PMAC fault display {Geo PMAC only}

Global Action Commands

<CTRL-A> – Abort all motion programs and moves
<CTRL-D> – Disable all PLC and PLCC programs
<CTRL-K> – Kill outputs for all motors
<CTRL-L> – Close rotary program buffer
<CTRL-O> – Do feed hold on all coordinate systems
<CTRL-Q> – Quit all programs at end of calculated moves
<CTRL-R> – Run working programs in all coordinate systems
<CTRL-S> – Step working programs in all coordinate systems
<CTRL-U> – Open rotary program buffer

Global Status Commands

<CTRL-B> – Report all motor status words to host
<CTRL-C> – Report all coordinate system status words to host
<CTRL-F> – Report all motor following errors (unscaled)
<CTRL-G> – Report global status words in binary form
<CTRL-P> – Report all motor positions (unscaled)
<CTRL-V> – Report all motor velocities (unscaled)
??? – Report global status words in hex ASCII
DATE – Report date of firmware version used
LIST [{buffer}] – Report contents of open [or specified] buffer
SIZE – Report size of open memory in words (sub-blocks)
TYPE – Report type of PMAC
VERSION – Report firmware revision level
EAVERSION – Report firmware revision level & information
CHECKSUM – Report firmware reference checksum value

Register Access Commands

R{address}[,{constant}] – Report contents of specified memory word address [or specified number of addresses] in decimal
RH{address}[,{constant}] – Report contents of specified memory word address [or specified number of addresses] in hex
W{address},{constant}[,{constant}..] – Write value to specified memory word address [or values to range]

PLC Control Commands

ENABLE PLC{constant}[,{constant}...] – Enable operation of specified interpreted PLC program[s]
DISABLE PLC{constant}[,{constant}...] – Disable operation of specified interpreted PLC program[s]
PAUSE PLC{constant}[,{constant}...] – Suspend operation of specified interpreted PLC program[s] at present point
RESUME PLC{constant}[,{constant}...] – Continue operation of specified interpreted PLC program[s] at paused point
ENABLE PLCC{constant}[,{constant}...] – Disable operation of specified compiled PLC program[s]
DISABLE PLCC{constant}[,{constant}...] – Disable operation of specified compiled PLC program[s]

Global Variable Commands

{constant} – Equivalent to **P0={constant}** if no unfilled table; otherwise value entered into table
I{constant}={expression} – Expression value assigned to I-variable
I{constant}[..{constant}]=* – Set I-variable[s] to default[s]
P{constant}[..{constant}]={expression} – Expression value assigned to P-variable(s)
M{constant}[..{constant}]={expression} – Expression value assigned to M-variable(s)
M{constant}->{definition} – M-variable defined as specified
M{constant}->* – M-variable defined as non-pointer variable
I{constant}[..{constant}] – Report I-variable value(s) to host
P{constant}[..{constant}] – Report P-variable value(s) to host
M{constant}[..{constant}] – Report M-variable value(s) to host
M{constant}[..{constant}]-> – Report M-variable definition(s) to host

Buffer Control Commands

OPEN PROG {constant} – Open specified motion program buffer for entering/editing
OPEN ROTARY – Open all defined rotary program buffer for entry
OPEN BINARY ROTARY – Open all defined rotary program buffers for binary entry thru DPRAM
OPEN PLC{constant} – Open specified PLC program buffer for entry
CLOSE – Close currently opened buffer
CLEAR – Erase contents of opened buffer
DEFINE GATHER [{constant}] – Set up a data gathering buffer using all open memory [or of specified size]
DELETE GATHER – erase the data gathering buffer
GATHER [TRIGGER] – Start data gathering [on external trigger]
ENDGATHER – Stop data gathering
DELETE PLCC{constant} – Erase specified compiled PLC program
DELETE TRACE – Erase the program trace buffer (no action taken; kept for backward compatibility)
DEFINE TBUF {constant} – Set up specified number of axis transformation matrices
DELETE TBUF – Erase all axis transformation matrices
DEFINE UBUFFER{constant} – Set up a user buffer of specified number of words

MACRO Ring Commands

MACROAUX – Report Type 0 MACRO slave variable value to host
MACROAUXREAD – Copy Type 0 MACRO slave variable to PMAC variable
MACROAUXWRITE – Copy PMAC variable value to Type 0 MACRO slave variable
MACROSLV {command} {node#} – Send command to Type 1 MACRO slave
MACROSLV {node#},{slave variable} – Report Type 1 MACRO slave variable value to host
MACROSLV {node#},{slave variable}={constant} – Set Type 1 MACRO slave variable value
MACROSLVREAD – Copy Type 1 MACRO slave variable to PMAC variable
MACROSLVWRITE – Copy PMAC variable value to Type 1 MACRO slave variable

On-line Coordinate System Commands

(These act immediately on currently addressed coordinate system)

Axis Definition Commands

#n->[{constant}]{axis}[+{constant}] – Define axis in terms of motor #, scale factor, and offset.

Examples: **#1->X**
#4->2000A+500

#n->[{constant}]{axis}[+[{constant}]{axis}[+[{constant}]{axis}]]
[+{constant}] – Define 2 or 3 axes in terms of motor #, scale factors, and offset. Valid only within XYZ or UVW groupings.

Examples: **#1->8660X-5000Y**
#2->5000X+8660Y+5000

#n-> – Report axis definition of Motor n in this C. S.
UNDEFINE – Erase definition of all axes in this C. S.

General Coordinate-System Commands

?? – Report coordinate system status in hex ASCII form
%{constant} – Specify feedrate override value
% – Report current feedrate override value to host

Program Control Commands

R – Run current program
S – Do one step of current program
B[{constant}] – Set program counter to specified location
H – Feed hold for coordinate system
A – Abort present program or move starting immediately
Q – Halt program; stop moves at end of last calculated program command
/ – Halt program execution at end of currently executing move
**** – Do program hold that permits jogging while in hold mode
MFLUSH – Erase contents of synchronous M-variable stack without executing

Coordinate-System Variable Commands

Q{constant} [..{constant}]={expression} – Assign expression value to Q-variable(s)
Q{constant} [..{constant}] – Report Q-variable value(s) to host

Axis Attribute Commands

{axis}={expression} – Change value of commanded axis position
Z – Make present commanded position of all axes in coordinate system equal to zero.
INC[({axis}[,{axis}...])] – Make all [or specified] axes do their moves incrementally
ABS[({axis}[,{axis}...])] – Make all [or specified] axes do their moves absolute
FRAX({axis}[,{axis}...]) – Make specified axes to be used in vector feedrate calculations
PMATCH – Re-match coordinate system axis positions to motor commanded positions
(used in case axis definition or motor position changed since last *axis* move)

Buffer Control Commands

PC – Report next program number and line (offset) to be executed to host
LIST PC[,[{constant}] – List next line of working program [and specified lines afterward] to be calculated
PE – Report working program number and line (offset) currently executing to host
LIST PE[,[{constant}] – List currently executing line of working program [and specified lines afterward]
DEFINE ROT {constant} – Establish rotary motion program buffer of specified word size for the addressed coordinate system
DELETE ROT – Erase rotary motion program buffer for addressed coordinate system
PR – Report number of lines between executing point and last loaded line in rotary program buffer.
LEARN – Read present commanded positions and add as axis commands to open program buffer

On-Line Motor Commands

(These act immediately on the currently-addressed motor. Except for the reporting commands, these commands are rejected if the motor is in a coordinate system that is currently running a motion program.)

General Motor Commands

$ – Reset motor – feedback device(s) and phasing
$* – Read absolute position of motor according to Ix10
HM – Perform homing routine for motor
HMZ – Perform zero-move homing routine for motor
SETPHASE – Set commutation angle for present position to Ix75
K – Kill output for motor
O{constant} – Open-loop output of specified magnitude

Jogging Commands

J+ – Jog motor indefinitely in positive direction
J- – Jog motor indefinitely in negative direction
J/ – Stop jogging motor; also restore to position control
J= – Jog motor to last pre-jog or pre-handwheel position
J={constant} – Jog motor to specified position
J=* – Variable jog to position
J:{constant} – Jog motor specified distance from current commanded position
J:* – Variable incremental jog from current commanded position
J^{constant} – Jog motor specified distance from current actual position
J^* – Variable incremental jog from current actual position
{jog command}^{constant} – Jog until trigger, final value specifies distance from trigger position to stop

Reporting Commands

P – Report position of motor
V – Report velocity of motor
F – Report following error of motor
? – Report status words for motor in hex ASCII form

Buffer Control Commands

DEFINE BLCOMP {entries},{count length} – Establish backlash compensation table for motor; to be filled by specified number of values

DELETE BLCOMP – Erase backlash compensation table for motor

DEFINE COMP {entries},[#{source},[#{target},]],{count length} – Establish leadscrew compensation table for motor; to be filled by specified number of values

DEFINE COMP{rows}.{columns},[#{source1},[#{source2},[#{target},]]],{row count length},{column count length} – Establish two dimensional leadscrew compensation table for motor; to be filled by specified number of values

DELETE COMP – Erase leadscrew compensation table for motor.

DEFINE TCOMP {entries},{count length} – Establish torque compensation table for motor; to be filled by specified number of values .

DELETE TCOMP – Erase torque compensation table for motor.

Motion Program Commands

Move Commands

{axis}{data}[{axis}{data}...] – Simple movement statement; can be used in LINEAR, RAPID or SPLINE modes

Example: **X1000 Y(P1) Z(P2*P3)**

{axis}{data}:{data}[{axis}{data}:{data}...] – Position:velocity move; to be used only in PVT mode

Example: **X5000:750 Y3500:(P3) A(P5+P6):100**

{axis}{data}^{data}[{axis}{data}^{data}...] – Move until trigger, variant of RAPID mode

{axis}{data}[{axis}{data}...][{vector}{data}...] – Circle move; to be used only in circular mode; vector is to circle center

Example: **X2000 Y3000 Z1000 I500 J300 K500**

DWELL{data} – Keep same commanded position; fixed time base

DELAY{data} – Keep same commanded position; variable time base

HOME{constant}[,{constant}...] – Home specified *motor(s)*

HOMEZ{constant}[,{constant}...] – Do zero-move homing of specified *motor(s)*

Move Mode Commands

LINEAR – Blended linear interpolation move mode
RAPID – Mode where all axes move a maximum velocity and accel.
CIRCLE1 – Clockwise circular interpolation move mode
CIRCLE2 – Counterclockwise circular interpolation move mode
PVT{data} – Position/velocity/time transition-point move mode (parabolic velocity profiles)
SPLINE1 – Uniform cubic spline move mode

SPLINE2 – Non-uniform cubic spline move mode
CC0 – Turns off cutter radius compensation
CC1 – Turns on cutter radius compensation left
CC2 – Turns on cutter radius compensation right

Axis Attribute Commands

ABS[({axis}[,{axis},...])] – Makes all [or specified] axes in absolute move mode
INC[({axis}[,{axis},...])] – Makes all [or specified] axes in incremental move mode
FRAX[({axis}[,{axis}...])] – Specifies feedrate axes
NORMAL{vector}{data}[{vector}{data}...] – Specifies normal vector to plane for circular moves and cutter compensation
PSET{axis}{data}[{axis}{data}...] – Sets axis position values
R{data} – Specifies circle radius; negative value is long arc
CCR{data} – Specifies cutter compensation radius value (modal)
TSEL{data} – Selects specified axis transformation matrix
TINIT – Initializes selected axis transformation matrix as identity matrix
ADIS{data} – Sets displacement vector of selected matrix to values starting with specified Q-variable
IDIS{data} – Increments displacement vector of selected matrix to values starting with specified Q-variable
AROT{data} – Sets rotation/scaling portion of selected matrix to values starting with specified Q-variable
IROT{data} – Incrementally changes rotation/scaling portion of selected matrix by multiplying it with values starting with specified Q-variable

Move Attribute Commands

TM{data} – Specifies move time (modal)
F{data} – Specifies move speed (modal)
TA{data} – Specifies move acceleration time (modal)
TS{data} – Specifies acceleration S-curve time (modal)

Variable Assignment Commands

I{constant}={expression} – Assigns expression value to specified I-variable
P{constant}={expression} – Assigns expression value to specified P-variable(s)
Q{constant}={expression} – Assigns expression value to specified Q-variable(s)
M{constant}={expression} – Assigns expression value to specified M-variable(s)
M{constant}=={expression} – Assigns expression synchronous with start of next move
M{constant}&={expression} – M-variable ANDed with expression synchronously
M{constant}|={expression} – M-variable ORed with expression synchronously
M{constant}^={expression} – M-variable XORed with expression synchronously

Program Logic Control

N{constant} – Line label
O{constant} – Alternate line label, stored as **N{constant}**
GOTO{data} – Jump to specified N-label; no return
GOSUB{data}[{letter}{axis}...] – Jump to specified N-label and return [with arguments]
CALL{data}[.{data}][{letter}{axis}...] – Jump to specified program [and label] [with arguments] and return.
RETURN – Return program operation to most recent **GOSUB** or **CALL**

READ ({letter} [,{letter}...]) – Allows subprogram or subroutine to take arguments
G{data} – **Gnn[.mmm]** interpreted as **CALL 1000.nnmmm**
(PROG 1000 provides subroutines for desired G-Code actions.)
M{data} – **Mnn[.mmm]** interpreted as **CALL 1001.nnmmm**
(PROG 1001 provides subroutines for desired M-Code actions.)
T{data} – **Tnn[.mmm]** interpreted as **CALL 1002.nnmmm**
(PROG 1002 provides subroutines for desired T-Code actions.)
D{data} – **Dnn[.mmm]** interpreted as **CALL 1003.nnmmm**
(PROG 1003 provides subroutines for desired D-Code actions.)
S{data} – Sets Q127 to value of **{data}**
PRELUDE1{call command} – For modal execution of call cmd. before subsequent moves
PRELUDE0 – De-activates modal **PRELUDE** calls
IF({condition}){action} – Conditionally execute action
IF({condition}) – Conditionally execute following statements
ELSE {action} – Execute action on previous false condition
ELSE – Execute following statements on previous false condition
ENDIF – Follows last of conditionally executed statements
WHILE({condition}) {action} – Do action as long as condition true
WHILE({condition}) – Do following statements as long as true
ENDWHILE – Follows last of conditionally executed statements
BLOCKSTART – So all commands until **BLOCKSTOP** to execute on Step
BLOCKSTOP – End of stepped statements starting on **BLOCKSTART**
STOP – Halts program execution; ready to resume
WAIT – Used with **WHILE** to halt execution while condition true

Miscellaneous Commands

COMMAND "{command}" – Issue command as if it came from host
COMMAND^{letter} – Issue control character command
SEND"{message}" – Transmit message over host interface
SENDS"{message}" – Transmit message over serial interface
SENDP"{message}" – Transmit message over parallel interface
DISPLAY [{constant}] "{message}" – Send message to LCD display [starting at specified location]
DISPLAY {constant},{constant}.{constant},{variable} – Send variable value to LCD using specified location and format
ENABLE PLC{constant}[,{constant}...] – Enable operation of specified interpreted PLC program[s]
DISABLE PLC{constant}[,{constant}...] – Disable operation of specified interpreted PLC program[s]
PAUSE PLC{constant}[,{constant}...] – Suspend operation of specified interpreted PLC program[s] at present point
RESUME PLC{constant}[,{constant}...] – Continue operation of specified interpreted PLC program[s] at paused point
ENABLE PLCC{constant}[,{constant}...] – Enable operation of specified compiled PLC program[s]
DISABLE PLCC{constant}[,{constant}...] – Disable operation of specified compiled PLC program[s]

PLC Program Commands

Conditions

IF ({condition}) – Evaluates condition to determine which branch to enter
WHILE ({condition}) – Conditional loop start; if true, holds up operation of PLC in the **WHILE** loop
AND ({condition}) – Forms compound condition w/ **IF** or **WHILE**
OR ({condition}) – Forms compound condition w/ **IF** or **WHILE**
ELSE – Starts false branch of **IF**
ENDIF – Closes out the actions dependent on an **IF** statement; used after, not before, an **ELSE** statement.
ENDWHILE – Closes out the actions dependent on a **WHILE** statement

Actions

{variable}={expression} – Expression value given to variable
MACROSLVREAD – Copy Type 1 MACRO slave variable to PMAC variable
MACROSLVWRITE – Copy PMAC variable value to Type 1 MACRO slave variable
COMMAND "{command}" – Issue command as if from host
COMMAND^{letter} – Issue control character command
SEND"{message}" – Send message to active host interface (serial or parallel)
SENDS"{message}" – Send message to serial interface
SENDP "{message}" – Send message to parallel (bus) interface
DISPLAY [{constant}] "{message}" – Display message on LCD display, starting at specified character
DISPLAY {constant}, {constant}.{constant}, {variable} – Send variable value to LCD using specified location and format.
ENABLE PLC{constant}[,{constant}...] – Enable operation of specified PLC program[s]
DISABLE PLC{constant}[,{constant}...] – Disable operation of specified PLC program[s]
PAUSE PLC{constant}[,{constant}...] – Suspend operation of specified interpreted PLC program[s] at present point
RESUME PLC{constant}[,{constant}...] – Continue operation of specified interpreted PLC program[s] at paused point
ENABLE PLCC{constant}[,{constant}...] – Enable operation of specified compiled PLC program[s]
DISABLE PLCC{constant}[,{constant}...] – Disable operation of specified compiled PLC program[s]

PMAC I-Variable Summary

General Divisions

I0 – I99 ..... General card setup (global)
I100 – I186 ..... Motor #1 setup
I187 – I199 ..... Coordinate System 1 setup
I200 – I286 ..... Motor #2 setup
I287 – I299 ..... Coordinate System 2 setup
...
I800 – I886 ..... Motor #8 setup
I887 – I899 ..... Coordinate System 8 setup
I900 – I979 ..... Encoder 1 - 16 setup (in groups of 5)
I980 – I1023 ...Reserved for future use

Global I-Variables

I0.....Serial Addressing Card Number {PMAC(1) w/Flex CPU, PMAC2 only}
I1.....Serial Port Communications Mode
I2.....Control Panel Disable
I3.....I/O Handshake Mode
I4.....Communications Checksum Enable
I5.....PLC Programs On/Off
I6.....Error Reporting Mode
I7.....In-Position Number of Cycles
I8.....Real Time Interrupt Period
I9.....Full/Abbreviated Program Listing Form
I10.....Servo Interrupt Time
I11.....Programmed Move Calculation Time
I12.....Jog-to-Position Calculation Time
I13.....Programmed Move Segmentation Time
I14.....Auto Position Match on Run
I15.....Degree/Radian Control For User Trig Functions
I16.....Rotary Buffer Request On Point
I17.....Rotary Buffer Request Off Point
I18.....Fixed Buffer Full Warning Point
I19.....Data Gathering Period (in Servo Cycles)
I20.....Data Gathering Selection Mask
I21–I44 .....Data Gathering Source 1–24 Address
I45.....Data Gathering Buffer Location and Mode
I46.....CPU Frequency Control {Flex CPU only}
I47.....Address of Pointer to **<CTRL-W>** Command
I48.....DPRAM Servo Data Enable
I49.....DPRAM Background Data Enable
I50.....Rapid Move Velocity Mode
I51.....Compensation Table Enable
I52.....'\' Program Hold Slew Rate
I53.....Program Step Mode Control
I54.....Serial Baud Rate Control {PMAC(1) w/Flex CPU, PMAC2 only}
I55.....DPRAM Background Buffer Control
I56.....DPRAM ASCII Communications Interrupt Enable
I57.....DPRAM Binary Rotary Buffer Enable
I58.....DPRAM ASCII Communications Enable

I59.....DPRAM Buffer Maximum Motor/C.S. Number
I60.....Auto-Converted ADC Register Address {PMAC(1) only}
I61.....Number of Auto-Converted ADC Registers {PMAC(1) only}
I62.....Internal Message Carriage Return Control
I63.....Control-X Echo Enable
I64.....Internal Response Tag Enable
I65.....User Configuration Variable
I66.....Servo-Channel ADC Auto-Copy Disable {PMAC2 only}
I67.....Modbus TCP Buffer Start Address
I68.....Alternate TWS Input Format
I69.....Modbus TCP Software Control Panel Start Address
I70-I77.....Analog Table Setup Lines {PMAC2 Only}
I8x.....Motor x Third-Resolver Gear Ratio
I89.....Cutter Comp Outside Angle Break Point
I90.....Minimum Arc Angle
I9x.....Motor x Second-Resolver Gear Ratio
I99.....Backlash Hysteresis

Motor I-Variables x = Motor Number (#x, x = 1 to 8}

Motor Definition I-Variables

Ix00.....Motor x Activate
Ix01.....Motor x PMAC-Commutate Enable
Ix02.....Motor x Command Output (DAC) Address
Ix03.....Motor x Position-Loop Feedback Address
Ix04.....Motor x Velocity-Loop Feedback Address
Ix05.....Motor x Master (Handwheel) Position Address
Ix06.....Motor x Master (Handwheel) Following Enable
Ix07.....Motor x Master (Handwheel) Scale Factor
Ix08.....Motor x Position-Loop Scale Factor
Ix09.....Motor x Velocity-Loop Scale Factor
Ix10.....Motor x Power-On Servo Position Address

Motor Safety I-Variables

Ix11.....Motor x Fatal (Shutdown) Following Error Limit
Ix12.....Motor x Warning Following Error Limit
Ix13.....Motor x Positive Software Position Limit
Ix14.....Motor x Negative Software Position Limit
Ix15.....Motor x Deceleration Rate on Position Limit or Abort
Ix16.....Motor x Maximum Permitted Program Velocity
Ix17.....Motor x Maximum Permitted Programm Accel.
Ix19.....Motor x Maximum Permitted Jog Accel.

Motor Movement I-Variables

Ix20.....Motor x (Jogging and Homing) Acceleration Time
Ix21.....Motor x (Jogging and Homing) S-Curve Time
Ix22.....Motor x Jog Speed
Ix23.....Motor x Homing Speed and Direction
Ix25.....Motor x Limit/Home Flag/Amp Flag Address
Ix26.....Motor x Home Offset
Ix27.....Motor x Position Rollover Range
Ix28.....Motor x In-position Band

Ix29.....Motor x Output - or First Phase – Bias

Motor Servo Control I-Variables {Standard PID Algorithm}

Ix30.....Motor x PID Proportional Gain
Ix31.....Motor x PID Derivative Gain
Ix32.....Motor x PID Velocity Feedforward Gain
Ix33.....Motor x PID Integral Gain
Ix34.....Motor x PID Integration Mode
Ix35.....Motor x PID Acceleration Feedforward Gain
Ix36.....Motor x Notch Filter Coefficient N1
Ix37.....Motor x Notch Filter Coefficient N2
Ix38.....Motor x Notch Filter Coefficient D1
Ix39.....Motor x Notch Filter Coefficient D2
Ix40.....Motor x Net Desired Position Filter Gain {Opt 6L Lookahead firmware only}
Ix57.....Motor x Continuous Current Limit
Ix58.....Motor x Integrated Current Fault Level
Ix59.....Motor x User-Written Servo Enable
Ix60.....Motor x Servo Cycle Extension Period
Ix61.....Motor x Current-Loop Integral Gain {PMAC2 only}
Ix62.....Motor x Forward-Path Current Loop Proportional Gain {PMAC2 only}
Ix63.....Motor x Integration Limit
Ix64.....Motor x "Deadband Gain Factor"
Ix65.....Motor x Deadband Size
Ix66.....Motor x PWM Scale Factor
Ix67.....Motor x Linear Position Error ("Big Step") Limit
Ix68.....Motor x Friction Feedforward Gain
Ix69.....Motor x Output Command (DAC) Limit

Motor Servo Control I-Variables {Option 6 Extended Servo Algorithm only}

Ix30.....Motor x ESA s0 Gain
Ix31.....Motor x ESA s1 Gain
Ix32.....Motor x ESA f0 Gain
Ix33.....Motor x ESA f1 Gain
Ix34.....Motor x ESA h0 Gain
Ix35.....Motor x ESA h1 Gain
Ix36.....Motor x ESA r1 Gain
Ix37.....Motor x ESA r2 Gain
Ix38.....Motor x ESA r3 Gain
Ix39.....Motor x ESA r4 Gain
Ix40.....Motor x ESA t0 Gain
Ix41.....Motor x ESA t1 Gain
Ix42.....Motor x ESA t2 Gain
Ix43.....Motor x ESA t3 Gain
Ix44.....Motor x ESA t4 Gain
Ix45.....Motor x ESA TS Gain
Ix46.....Motor x ESA L1 Gain
Ix47.....Motor x ESA L2 Gain
Ix48.....Motor x ESA L3 Gain
Ix49.....Motor x ESA k0 Gain
Ix50.....Motor x ESA k1 Gain
Ix51.....Motor x ESA k2 Gain

Ix52.....Motor x ESA k3 Gain
Ix53.....Motor x ESA KS Gain
Ix54.....Motor x ESA d1 Gain
Ix55.....Motor x ESA d2 Gain
Ix56.....Motor x ESA g0 Gain
Ix57.....Motor x ESA g1 Gain
Ix58.....Motor x ESA GS Gain
Ix60.....Motor x Servo Cycle Extension Period
Ix61.....Motor x Current-Loop Integral Gain {PMAC2 only}
Ix62.....Motor x Forward-Path Current Loop Proportional Gain {PMAC2 only}
Ix68.....Motor x Friction Feedforward Gain
Ix69.....Motor x Output Command (DAC) Scale Factor

Motor Commutation I-Variables

Ix70.....Motor x Number of Commutation Cycles (N) for Cycle Size Definition
Ix71.....Motor x Encoder Counts per N Commutation Cycles
Ix72.....Motor x Commutation Phase Angle
Ix73.....Motor x Phase Finding Output (DAC) Value
Ix74.....Motor x Phase Finding Time
Ix75.....Motor x Phasing Offset
Ix76.....Motor x Velocity Phase Advance Gain {PMAC(1) only}
Ix76.....Motor x Current-Loop Back-Path Proportional Gain {PMAC2 only}
Ix77.....Motor x Induction Motor Magnetization Current
Ix78.....Motor x Induction Motor Slip Gain
Ix79.....Motor x Second Phase Bias
Ix80.....Motor x Power-Up Mode
Ix81.....Motor x Power-On Phase Position Address
Ix82.....Motor x Current Loop Feedback Address {PMAC2 only}
Ix83.....Motor x Ongoing Phasing Position Address
Ix84.....Motor x Current Loop Mask Word {PMAC2 only}

Further Motor I-Variables

Ix85.....Motor x Backlash Take-up Rate
Ix86.....Motor x Backlash Size

Coordinate System I-Variables

x = Coordinate System Number (&x, x = 1 to 8)

Ix87.....Coordinate System x Default Program Acceleration Time
Ix88.....Coordinate System x Default Program S-Curve Time
Ix89.....Coordinate System x Default Program Feedrate
Ix90.....Coordinate System x Feedrate Time Units
Ix91.....Coordinate System x Default Working Program Number
Ix92.....Coordinate System x Move Blend Disable
Ix93.....Coordinate System x Time Base Control Register Address
Ix94.....Coordinate System x Time Base Slew Rate
Ix95.....Coordinate System x Feed Hold Slew Rate
Ix96.....Coordinate System x Maximum Circle Error Limit
Ix98.....Coordinate System x Maximum Feedrate

PMAC(1) Servo Interface Setup I-Variables

For Encoder n (n = 1 to 16)

I900 - I904 – Encoder 1
I905 - I909 – Encoder 2
I910 - I914 – Encoder 3
I915 - I919 – Encoder 4
...
I970 - I974 – Encoder 15
I975 - I979 – Encoder 16

I900, I905, I910, I915, I920, I925, I930, I935, I940, I945, I950, I955, I960, I965, I970, I975	(Encoder I-Variable 0)	Encoder n Decode Control
I901, I906, I911, I916, I921, I926, I931, I936, I941, I946, I951, I956, I961, I966, I971, I976	(Encoder I-Variable 1)	Encoder n Filter Disable
I902, I907, I912, I917, I922, I927, I932, I937, I942, I947, I952, I957, I962, I967, I972, I977	(Encoder I-Variable 2)	Encoder n Position Capture Control
I903, I908, I913, I918, I923, I928, I933, I938, I943, I948, I953, I958, I963, I968, I973, I978	(Encoder I-Variable 3)	Encoder n Flag Select Control

PMAC2 Servo Interface Setup I-Variables

Global Hardware Setup I-Variables

I900.....MaxPhase and PWM 1-4 Frequency Control
I901.....Phase Clock Frequency Control
I902.....Servo Clock Frequency Control
I903.....Hardware Clock 1-4 Frequency Control
I904.....PWM 1-4 Deadtime/PFM 1-4 Pulse-Width Control
I905.....DAC 1-4 Strobe Word Control
I906.....PWM 5-8 Frequency Control
I907.....Hardware Clock 5-8 Frequency Control
I908.....PWM 5-8 / PFM 5-8 Pulse-Width Control
I909.....DAC 5-8 Strobe Word Control

Channel n Hardware Setup I-Variables

I9n0.....Encoder/Timer n Decode Control
I9n1.....Position Compare n Channel Select
I9n2.....Encoder n Capture Control
I9n3.....Flag n Capture Select
I9n4.....Encoder n Gated Index Select
I9n5.....Encoder n Index Gate State
I9n6.....Output n Mode Select
I9n7.....Output n Invert Control
I9n8.....PFM n Direction Invert Control
I9n9.....Encoder n Hardware 1/T Control

Ultra-Lite/Supplemental Channel Hardware Setup I-Variables

I990.....Handwheel 1 Decode Control
I991.....Handwheel 2 Decode Control
I992.....Ultralite MaxPhase Frequency Control

I993.....Supplemental Hardware Clock Control
I994.....Supplemental Deadtime/Pulse-Width Control
I995.....MACRO Ring Master/Slave Control
I996.....MACRO Node Activation Control
I997.....Ultralite Phase Clock Frequency Control
I998.....Ultralite Servo Clock Frequency Control

MACRO Support I-Variables

I1000.....MACRO Node Auxiliary Register Enable
I1001.....MACRO Ring Check Time Period
I1002.....MACRO Node Protocol Type Control
I1003.....MACRO Type 1 Master/Slave Communications Timeout
I1004.....MACRO Ring Error Shutdown Count
I1005.....MACRO Ring Sync Packet Shutdown Count
I1010.....Resolver Excitation Phase Offset
I1011.....Resolver Excitation Gain
I1012.....Resolver Excitation Frequency Divider
I1013.....Motor Temperature Check Enable
I1015.....SSI Clock Frequency Control
I1016.....SSI Channel 1 Mode Control
I1017.....SSI Channel 1 Word Length Control
I1018.....SSI Channel 2 Mode Control
I1019.....SSI Channel 2 Word Length Control
I1020.....Lookahead Length (Option 6L firmware only)
I1021.....Lookahead State Control (Option 6L firmware only)

PMAC Error Code Summary

PMAC can report the following error messages in response to commands:

Error	Problem	Solution
ERR001	Command not allowed during program execution	(should halt program execution before issuing command)
ERR002	Password error	(should enter the proper password)
ERR003	Data error or unrecognized command	(should correct syntax of command)
ERR004	Illegal character: bad value (>127 ASCII) or serial parity/framing error	(should correct the character and or check for noise on the serial cable)
ERR005	Command not allowed unless buffer is open	(should open a buffer first)
ERR006	No room in buffer for command	(should allow more room for buffer – **DELETE** or **CLEAR** other buffers)
ERR007	Buffer already in use	(should **CLOSE** currently open buffer first)
ERR008	MACRO auxiliary communications error	(should check MACRO ring hardware and software setup)
ERR009	Program structural error (e.g. **ENDIF** without **IF**)	(should correct structure of program)
ERR010	Both overtravel limits set for a motor in the C.S.	(should correct or disable limits)
ERR011	Previous move not completed	(should **Abort** it or allow it to complete)
ERR012	A motor in the coordinate system is open-loop	(should close the loop on the motor)
ERR013	A motor in the coordinate system is not activated	(should set Ix00 to 1 or remove motor from C.S.)
ERR014	No motors in the coordinate system	(should define at least one motor in C.S.)
ERR015	Not pointing to valid program buffer	(should use **B** command first, or clear out scrambled buffers)
ERR016	Running improperly structured program (e.g. missing **ENDWHILE**)	(should correct structure of program)
ERR017	Trying to resume after **/** or **** with motors out of stopped position	(should use **J=** to return motor[s] to stopped position)

.

Note

Variable I6 controls whether and how these error messages are sent

PMAC Syntax Notes

1. PMAC syntax is not case-sensitive. That is, it does not matter whether an upper-case or lower-case letter is used in any command or statement. PMAC commands are shown in this document in all upper-case letters to help distinguish them better from the explanatory text.

   Example: **X1000** and **x1000** are identical statements to PMAC.

2. In syntax definitions, an item in squiggly brackets, such as **{data}**, means you can put what you wish into that part of the syntax, subject to the defined limitations of that item.

   Example: If the syntax is **X{data}**, you can put **X1000**, **X(P1)**, or **X(P2*P3+50)**, because **1000**, **(P1)**, and **(P2*P3+50)** all fit in the defined limitations for **{data}**.

3. In syntax definitions, items contained within square brackets are optional to the syntax. If there is an ellipsis (...) within the square brackets, items contained within the square brackets can be repeated.

   Example: If the syntax definition is **{axis}{data} [{axis}{data} ...]**, you can put **X1000**, **X1000Y1000**, or **X1000Y1000Z1000**.

4. Spaces are not important in PMAC syntax, except where specifically noted.

PMAC I-VARIABLE SPECIFICATION

On PMAC, I-variables (Initialization, or Set-up, Variables) determine the "personality" of the controller for a given application. They are at fixed locations in memory and have pre-defined meanings. Most are integer values, and their range varies depending on the particular variable. There are 1024 I- variables, from I0 to I1023, and they are organized as follows:

I-Variable	Function
I0 – I99	General card setup (global)
I100 – I186	Motor #1 setup
I187 – I199	Coordinate System 1 setup
I200 – I286	Motor #2 setup
I287 – I299	Coordinate System 2 setup
…	
I800 – I886	Motor #8 setup
I887 – I899	Coordinate System 8 setup
I900 – I999	Hardware Channel setup
I1000 – I1023	MACRO and reserved

Global I-Variables

I0 Serial Addressing Card Number {PMAC(1) w/Flex CPU, PMAC2 only}

Range $0 to $F (0 to 15)

Units none

Default 0

Remarks I0 controls the card number for software addressing purposes on a multi-drop serial communications cable for all PMAC2 boards and for PMAC(1) boards with an Option 5xF "Flex" CPU. (On other PMAC(1) boards, the card number is determined by the settings of jumpers E40 – E43.)

If I2 is set to 2, the PMAC must be addressed with the `@n` command, where `n` matches the value of I0 on the board, before it will respond. If the PMAC receives the `@n` command, where `n` does not match I0 on the board, it will stop responding to commands on the serial port. No two boards on the same serial cable may have the same value of I0.

If the `@@` command is sent over the serial port, all boards on the cable will respond to action commands. However, only the board with I0 set to 0 will respond to the host with handshake characters and/or data responses. All boards on the cable will respond to control-character action commands such as `<CTRL-R>`, regardless of the current addressing.

Note:

RS-422 serial interfaces must be used on all PMAC boards for multi-drop serial communications; this will not work with RS-232 interfaces. If the RS-422 interface is not present as a standard feature on the PMAC board, the Option 9L serial converter module must be purchased. It is possible to use an RS-232 interface on the host computer, connected to the RS-422 ports on the PMAC2 boards.

Typically multiple PMAC2 boards on the same serial cable will share servo and phase clock signals over the serial port cable for tight synchronization. If the servo and phase clock lines are connected between multiple PMACs, only one of the PMAC boards can be set up to output these clocks (E40 – E43 all ON for a PMAC(1), E1 jumper OFF for a PMAC2). All of the other boards in the chain must be set up to input these clocks (one or more of the jumpers E40 – E43 OFF for a PMAC(1), E1 jumper ON for a PMAC2).

Note:

Any PMAC(1) board with one or more of E40 – E43 OFF, or any PMAC2 board with jumper E1 ON, is expecting its SERVO and PHASE clock signals externally from a Card 0. If it does not receive these clock signals, the watchdog timer will immediately shut down the board and the red LED will light.

If the PMAC2 has E1 ON to receive external SERVO and PHASE clock signals for synchronization purposes, but is not using multi-drop serial communications, I0 does not need to be changed from 0.

To set up a board to communicate as Card 1 to Card 15 on a multi-drop serial cable, first communicate with the board as Card 0. Set I0 to specify the card number (software address) that the board will have on the multi-drop cable. Also, set I1 to 2 to enable the serial software addressing. Store these values to the non-volatile flash memory with the `SAVE` command. Then turn off power; if the board is to input its clocks, put a jumper on E1; connect the multi-drop cable; restore power to the system.

I1 Serial Port Mode

Range 0 .. 3

Units none

Default 0

Remarks I1 controls two aspects of how PMAC uses its serial port. The first aspect is whether PMAC uses the CS (CTS) handshake line to decide if it can send a character out the serial port. The second aspect is whether PMAC will require software card addressing, permitting multiple cards to be daisychained on a single serial line.

There are four possible values of I1, covering all the possible combinations:

Setting	Meaning
0	CS handshake used; no software card address required
1	CS handshake not used; no software card address required
2	CS handshake used; software card address required
3	CS handshake not used; software card address required

When CS handshaking is used (I1 is 0 or 2), PMAC waits for CS to go true before it will send a character. This is the normal setting for real serial communications to a host; it allows the host to hold off PMAC messages until it is ready.

When CS handshaking is not used (I1 is 1 or 3), PMAC disregards the state of the CS input and always sends the character immediately. This mode permits PMAC to "output" messages, values, and acknowledgments over the serial port even when there is nothing connected, which can be valuable in stand-alone and PLC-based applications where there are **`SEND`** and **`CMD`** statements in the program. If these strings cannot be sent out the serial port, they can "back up", stopping program execution.

When software addressing is not used (I1 is 0 or 1), PMAC assumes that it is the only card on the serial line, so it always acts on received commands, sending responses back over the line as appropriate.

When software addressing is used (I1 is 2 or 3), PMAC assumes that there are other cards on the line, so it requires that it be addressed (with the **@{card** command) before it responds to commands. The **{card}** number in the command must match the card number set up in hardware on the card with jumpers or DIP-switches.

See Also Serial Port, Multiple-Card Applications (Talking to PMAC)
I-variable I6
Program Commands **SEND**, **CMD**
Connectors J4 (PMAC-PC, -Lite, -VME), J1, J3 (PMAC-STD)
Jumpers E40-E43 (PMAC-PC, -Lite, -VME)
DIP-switches SW1-1 – SW1-4 (PMAC-STD)

I2 Control Panel Disable

Range 0 .. 3

Units none

Default 0

Remarks I2 allows the enabling and disabling of the control panel discrete inputs (on the JPAN connector). I2=0 enables these control panel functions; I2=1 disables them. When disabled, these inputs can be used as general purpose I/O. The reset, handwheel, and wiper inputs on the JPAN connector are not affected by I2.

When I2=0, the IPOS, EROR and F1ER status lines to JPAN and the Programmable Interrupt Controller (PIC), and the BREQ status line to the PIC, reflect the hardware-selected coordinate system (by BCD-coded lines FPDn/ on JPAN); when I2=1, they reflect the software-addressed coordinate system (&n).

When I2=3, the discrete inputs on the JPAN connector are disabled, and the dual-ported RAM control panel functions are enabled. Refer to the descriptions of DPRAM functions for more detail.

See Also Using Interrupts (Writing a Host Communications Program)
I-variables I16-I18
Custom Inputs Example (JOGSWTCH.PMC)
Connector JPAN (J2)
DPRAM Control Panel Functions

I3 I/O Handshake Control

Range 0 .. 3

Units none

Default 1

Remarks I3 controls what characters, if any, are used by PMAC to delimit a transmitted line, and whether PMAC issues an acknowledgment (handshake) of a command.

Note:

With checksum enabled (I4=1), checksum bytes are added after the handshake character bytes.

Valid values of I3 and the modes they represent are:

0: PMAC does not acknowledge receipt of a valid command. It returns a **<BELL>** character on receipt of an invalid command. Messages are sent without beginning or terminating **<LF>** (line feed); simply as **DATA <CR>** (carriage return).

1: PMAC acknowledges receipt of a valid **<CR>**-terminated command with a **<LF>**; of an invalid command with a **<BELL>** character. Messages are sent as **<LF> DATA <CR> [<LF> DATA <CR> ...] <LF>**. (The final **<LF>** is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [**SEND** or **CMD**]). This setting is good for communicating with terminal display programs, such as the PMAC Executive program.

2: PMAC acknowledges receipt of a valid **<CR>**-terminated command with an **<ACK>**; of an invalid command with a **<BELL>** character. Messages are sent as **DATA <CR> [DATA <CR> ...] <ACK>**. (The final **<ACK>** is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [**SEND** or **CMD**]). This is probably the best setting for fast communications with a host program without terminal display.

3: PMAC acknowledges receipt of a valid **<CR>**-terminated command with an **<ACK>**; of an invalid command with a **<BELL>** character. Messages are sent as **<LF> DATA <CR> [<LF> DATA <CR> ...] <ACK>**. (The final **<ACK>** is the acknowledgment of the host command; it does not get sent with a message initiated from a PMAC program [**SEND** or **CMD**]).

Note:

When I58=1 to enable DPRAM ASCII communications, I3 is forced to 0 or 2 from 1 or 3, respectively.

Example

With I3=0:

```
#1J+<CR>.....                    ; Valid command not requiring data response
.....                            ; No acknowledging character
UUU<CR> .....                    ; Invalid command
<BELL>.....                      ; PMAC reports error
P1..3<CR> .....                  ; Valid command requiring data response
25<CR>50<CR>75<CR>               ; PMAC responds with requested data
```

With I3=1:

```
#1J+<CR>.....                    ; Valid command not requiring data response
<LF> .....                       ; Acknowledging character
UUU<CR> .....                    ; Invalid command
<BELL>.....                      ; PMAC reports error
P1..3<CR> .....                  ; Valid command requiring data response
<LF>25<CR><LF>50<CR><LF>75<CR><LF>
.....                            ; PMAC responds with requested data
```

With I3=2:

```
#1J+<CR>.....                    ; Valid command not requiring data response
<ACK>......                      ; Acknowledging character
UUU<CR> .....                    ; Invalid command
<BELL>.....                      ; PMAC reports error
P1..3<CR> .....                  ; Valid command requiring data response
25<CR>50<CR>75<CR><ACK>
```

```
.....                                   ; PMAC responds with requested data
With I3=3:
#1J+<CR>.....                           ; Valid command not requiring data response
<ACK>.....                              ; Acknowledging character
UUU<CR> .....                           ; Invalid command
<BELL>.....                             ; PMAC reports error
P1..3<CR> .....                         ; Valid command requiring data response
<LF>25<CR><LF>50<CR><LF>75<CR><ACK>
.....                                   ; PMAC responds with requested data
```

See Also Talking to PMAC
Writing a Host Communications Program
I-variables I4, I6, I58

I4 Communications Integrity Mode

Range 0 .. 3

Units none

Default 0

Remarks I4 permits PMAC to compute checksums of the communications bytes (characters) sent either way between the host and PMAC, and also controls how PMAC reacts to serial character errors (parity and framing), if found. Parity checking is only enabled if jumper E49 is OFF for PMAC-PC, -Lite, -VME; or ON for PMAC-STD.

The possible settings of I4 are:

Setting	Meaning
0	Checksum disabled, serial errors reported immediately
1	Checksum enabled, serial errors reported immediately
2	Checksum disabled, serial errors reported at end of line
3	Checksum enabled, serial errors reported at end of line

Communications Checksum: With I4=1 or 3, PMAC computes the checksum for communications in either direction and sends the checksum to the host. It is up to the host to do the comparison between PMAC's checksum and the checksum it computed itself. PMAC does not do this comparison. The host should never send a checksum byte to PMAC.

Host-to-PMAC Checksum: PMAC will compute the checksum of a communications line sent from the host to PMAC. The checksum does not include any control characters sent (not even the final Carriage-Return). The checksum is sent to the host immediately following the acknowledging handshake character (**<LF>** or **<ACK>**), if any. Note that this acknowledging and handshake comes after any data response to the command (and its checksum!). If PMAC detects an error in the line through its normal syntax checking, it will respond with the **<BELL>** character, but will not follow this with a checksum byte.

Note:

The on-line command **<CTRL-N>** can be used to verify the checksum of a command line before the **<CR>** has been sent. The use of **<CTRL-N>** does not affect how I4 causes PMAC to report a checksum after the **<CR>** has been sent.

PMAC-to-Host Checksum: PMAC will compute the checksum of any communications line it sends to the host. This checksum includes control characters sent with the line, including the final **<carriage-return>**. The checksum is sent immediately following this **<carriage-return>**. On a multiple-line response, one checksum is sent for each line. Note that this checksum is sent before the checksum of the command line that caused the response.

For more details on checksum, refer to the Writing a Host Communications Program section of the manual.

Serial character errors: If PMAC detects a serial character error, it will set a flag so that the entire command line will be rejected as having a syntax error after the **<CR>** is sent. With I4=0 or 1, it will also send a **<BELL>** character to the host immediately on detecting the character error. Note that this mode will catch a character error on the **<CR>** as well, whereas in the I4=2 or 3 mode, the host would have to catch an error on the **<CR>** character by the fact that PMAC would not respond (because it never saw a **<CR>**).

See Also Communications Checksum (Writing a Host Communications Program)
I-variables I3, I6
On-line command **<CTRL-N>**
Jumper E49

I5 PLC Programs On/Off

Range 0 .. 3

Units none

Default 0

Remarks I5 controls which PLC programs may be enabled. There are two types of PLC programs: the foreground programs (interpreted PLC 0 and compiled PLCC 0), which operate at the end of servo interrupt calculations, with a repetition rate determined by I8 (PLC 0 should be used only for time-critical tasks and should be short); and the background programs (interpreted PLC 1 to PLC 31 and compiled PLCC1 to PLCC 31) which cycle repeatedly in background as time allows. I5 controls these as follows:

Setting	Meaning
0	Foreground PLCs off; background PLCs off
1	Foreground PLCs on; background PLCs off
2	Foreground PLCs off; background PLCs on
3	Foreground PLCs on; background PLCs on

Note that an individual PLC program still needs to be enabled to run – a proper value of I5 merely permits it to be run. Any PLC program that exists at power-up or reset is automatically enabled (even if the saved value of I5 does not permit it to run immediately); also, the **ENABLE PLC n** or **ENABLE PLCC n** command enables the specified program(s). A PLC program is disabled either by the **DISABLE PLC n** or **DISABLE PLCC n** command, or by the **OPEN PLC n** command. A **CLOSE** command does not automatically re-enable an interpreted PLC program – it must be done explicitly. When the compiled code for PLCC programs is downloaded to the PMAC, they are automatically enabled if permitted by I5.

See Also Running PLC Programs (Writing a PLC Program)
On line commands **ENABLE PLC n**, **DISABLE PLC n**, **ENABLE PLCC n**, **DISABLE PLCC n**, **OPEN PLC n**, **CLOSE**, **<CTRL-D>**, **$$$.**

I6 Error Reporting Mode

Range 0 .. 3

Units none

Default 3

Remarks I6 controls how PMAC reports errors in command lines. When I6 is set to 0 or 2, PMAC reports any error only with a **<BELL>** character. When I6 is 0, the **<BELL>** character is given for invalid commands issued both from the host and from PMAC programs (using **CMD"{command}"**). When I6 is 2, the **<BELL>** character is given only for invalid commands from the host; there is no response to invalid commands issued from PMAC programs. (In no mode is there a response to valid commands issued from PMAC programs.)

When I6 is set to 1 or 3, an error number message can be reported along with the **<BELL>** character. The message comes in the form of **ERRnnn<CR>**, where **nnn** represents the three-digit error number. If I3 is set to 1 or 3, there is a **<LF>** character in front of the message.

When I6 is set to 1, the form of the error message is **<BELL>{error message}**. This setting is the best for interfacing with host-computer driver routines. When I6 is set to 3, the form of the error message is **<BELL><CR>{error message}**. This setting is appropriate for use with the PMAC Executive Program in terminal mode.

Currently, the following error messages can be reported:

Error	Problem	Solution
ERR001	Command not allowed during program execution	(should halt program execution before issuing command)
ERR002	Password error	(should enter the proper password)
ERR003	Data error or unrecognized command	(should correct syntax of command)
ERR004	Illegal character: bad value (>127 ASCII) or serial parity/framing error	(should correct the character and or check for noise on the serial cable)
ERR005	Command not allowed unless buffer is open	(should open a buffer first)
ERR006	No room in buffer for command	(should allow more room for buffer – **DELETE** or **CLEAR** other buffers)
ERR007	Buffer already in use	(should **CLOSE** currently open buffer first)
ERR008	MACRO ring auxiliary communications error	(should correct MACRO communications or connections)
ERR009	Program structural error (e.g. **ENDIF** without **IF**)	(should correct structure of program)
ERR010	Both overtravel limits set for a motor in the C.S.	(should correct or disable limits)
ERR011	Previous move not completed	(should **Abort** it or allow it to complete)
ERR012	A motor in the coordinate system is open-loop	(should close the loop on the motor)
ERR013	A motor in the coordinate system is not activated	(should set Ix00 to 1 or remove motor from C.S.)
ERR014	No motors in the coordinate system	(should define at least one motor in C.S.)
ERR015	Not pointing to valid program buffer	(should use **B** command first, or clear out scrambled buffers)
ERR016	Running improperly structured program (e.g. missing **ENDWHILE**)	(should correct structure of program)

ERR017	Motor(s) in C.S. not at halted position to restart after **/** or **** command	(should move motor(s) back to halted position with **J=**)

See Also Talking to PMAC
Writing a Host Communications Program
I-variables I3, I4
On-line commands **R**, **S**

I7 In-Position Number of Cycles

Range 0 .. 255

Units Background computation cycles (minus one)

Default 0

Remarks I7 permits the user to define the number of consecutive scans that PMAC motors must satisfy all "in-position" conditions before the motor in-position bit is set true. This permits the user to ensure that the motor is truly settled in the end position before executing the next operation, on or off PMAC. I7 + 1 consecutive scans are required.

PMAC scans for the in-position condition of each active motor during the "housekeeping" part of every background cycle, which occurs between each scan of each enabled uncompiled background PLC (PLC 1-31). All motors in a coordinate system must have true in-position bits for the coordinate-system in-position bit to be set true.

See Also Control Panel Port (Connecting PMAC to the Machine)
Using Interrupts (Writing a Host Communications Program)
I-variable Ix28
On-line commands **?**,**??**
Suggested M-variable definitions Mx40
Memory registers Y:$0814, Y:$08D4, etc., Y:$0817, Y:$08D7, etc.
DPRAM Control Panel Functions
JPAN connector

I8 Real Time Interrupt Period

Range 0 .. 255

Units Servo Interrupt Cycles

Default 2

Remarks I8 controls how often certain time-critical tasks, such as PLC 0 and checking for motion program move planning, are performed. A value of 2 means that they are performed after every third servo interrupt, 3 means every fourth interrupt, and so on. The vast majority of users can leave this at the default value. In some advanced applications that push PMAC's speed capabilities, tradeoffs between performance of these tasks and the calculation time they take may have to be evaluated in setting this parameter.

Note:

A large PLC 0 with a small value of I8 can cause severe problems, because PMAC will attempt to execute the PLC program every I8 cycle. This can starve background tasks, including communications, background PLCs, and even updating of the watchdog timer, for time, leading to erratic performance or possibly even shutdown.

In multiple-card PMAC applications where it is very important that motion programs on the two cards start as closely together as possible, I8 should be set to 0. In this case, no PLC 0 should be running when the cards are awaiting a Run command. At other times I8 may be set greater than 0 and PLC 0 re-enabled.

See Also How PMAC Executes a Motion Program (Writing a Motion Program)
PLC 0 (Writing a PLC Program)

I9 Full/Abbreviated Program Listing Form

Range 0 .. 3

Units none

Default 2

Remarks I9 controls aspects of how PMAC reports program listings and variable values. The following table shows the values of I9 and what they represent:

Setting	Meaning
0	Short form, decimal address I-variable return
1	Long form, decimal address I-variable return
2	Short form, hex address I-variable return
3	Long form, hex address I-variable return

When this parameter is 0 or 2, programs are sent back in abbreviated form for maximum compactness, and when I-variable values or M-variable definitions are requested, only the values or definitions are returned, not the full statements. When this parameter is 1 or 3, programs are sent back in full form for maximum readability. Also, I-variable values and M-variable definitions are returned as full command statements, which is useful for archiving and later downloading.

When this parameter is 0 or 1, I-variable values that specify PMAC addresses are returned in decimal form. When it is 2 or 3, these values are returned in hexadecimal form (with the '$' prefix). You are always free to send any I-variable values to PMAC either in hex or decimal, regardless of the I9 setting. This does not affect how I-variable assignment statements inside PMAC motion and PLC programs are reported when the program is listed.

Example With I9=0:

```
I125 .....                ; Request address I-variable value
49152.....                ; PMAC reports just value, in decimal
M101->.....               ; Request M-variable definition
X:$C001,24,S              ; PMAC reports just definition
LIST PROG 1..             ; Request listing of program
LIN.....                  ; PMAC reports program short form
X10
DWE1000
RET
```

With I9=1:

```
I125 .....                ; Request address I-variable value
I125=49152....            ; PMAC reports whole statement, in decimal
M101->.....               ; Request M-variable definition
M101->X:$C001,24,S        ; PMAC reports whole statement
LIST PROG 1..             ; Request listing of program
LINEAR.....               ; PMAC reports program long form
X10
```

```
DWELL1000
RETURN
```

With I9=2:

```
I125 .....                ; Request address I-variable value
$C000.....                ; PMAC reports just value, in hexadecimal
```

With I9=3:

```
I125 .....                ; Request address I-variable value
I125=$C000                ; PMAC reports whole statement, in hexadecimal
```

See Also Talking to PMAC
On-line commands **I{constant}**, **M{constant}->**, **LIST**
I-Variables I19-I44, I47, Ix02-Ix05, Ix25, Ix83, Ix93

I10 Servo Interrupt Time

Range 0 .. 8,388,607

Units 1 / 8,388,608 msec

Default 3,713,707

Remarks I10 tells PMAC how much time there is between servo interrupts (which is controlled by hardware circuitry), so that the interpolation software knows how much time to increment each servo interrupt.

The fundamental equation for I10 is:

$$I10 = \frac{8{,}388{,}608}{ServoFrequency(kHz)} = 8{,}388{,}608 * ServoTime(m\,sec)$$

On PMAC(1), the servo interrupt time is determined by the settings of hardware jumpers E98, E29-E33, and E3-E6. The proper value of I10 can be determined from the settings of these jumpers by the formula:

$$I10 = 232{,}107 * E98JumperFactor * PhaseJumperFactor * ServoJumperFactor$$

where the factors can be taken from the following:

E98 Setting	1-2	2-3
E98JumperFactor	1	2

Phase Jumper ON	E29	E30	E31	E32	E33
PhaseJumperFactor	16	8	4	2	1

$$ServoJumperFactor = 1 + E3 + (2 * E4) + (4 * E5) + (8 * E6)$$

in which *En* = 0 if the jumper is ON, and En = 1 if the jumper is OFF.

On PMAC2, the servo interrupt time is determined on PMAC2 Ultralite boards by MACRO IC I-variables I992, I997, and I998; on non-Ultralite boards by Servo IC I-variables I900, I901, and I902; The proper setting of I10 can be determined from Servo IC variables by the formula:

$$I10 = \frac{640}{9}(2 * I900 + 3)(I901 + 1)(I902 + 1)$$

The proper setting of I10 can be determined from MACRO IC variables by the formula:

$$I10 = \frac{640}{9}(2 * I992 + 3)(I997 + 1)(I998 + 1)$$

I10 is used to provide the "delta-time" value in the position update calculations, scaled such that 2^{23} – 8,388,608 – means one millisecond. Delta-time in these equations is I10*(%value/100). The % (feedrate override) value can be controlled in any of several ways: with the on-line '%' command, with a direct write to the command '%' register, with an analog voltage input, or with a digital input frequency. The default % value is 100, and many applications can always leave it at 100.

Note:

Even if Ix60 (servo cycle extension) has been changed from its default value of 0 for any or all motors, the value of I10 should reflect the time between servo *interrupts*, not between consecutive servo cycle calculations.

See Also Setting the Servo Update Time (Servo Features)
Jumpers E3-E6, E29-E33, E98
Connector J4 Pins 21-24 (PMAC-PC, -VME), J4 Pins 1 & 8 (PMAC-Lite), J3 Pins 5-8 (PMAC-STD).

I11 Programmed Move Calculation Time

Range 0 .. 8,388,607

Units msec

Default 0

Remarks I11 controls the delay from when the run signal is taken (or the move sent if executing immediately) and when the first programmed move starts. If several PMACs need to be run synchronously, I11 should be set the same on all of the cards. If I11 is set to zero, the first programmed move starts as soon as the calculation is complete.

This calculation time delay is also used after any break in the continuous motion of a motion program: a **DWELL**, a **PSET**, a **WAIT**, or each move if Ix92=1 (a **DELAY** is technically a zero-distance move, and so does not constitute a break).

The actual delay time varies with the time base (e.g. at a value of 50, the actual delay time will be twice the number defined here), which keeps it as a fixed *distance* of the master in an external time base application. If it is desired to have the slave coordinate system start up immediately with the master, I11 should be set to zero, and the program commanded to run *before* the master starts to move.

Note:

If I11 is greater than zero, defining a definite time for calculations, and PMAC cannot complete the calculations for the first move of a sequence by the end of the I11 time, PMAC will terminate the running of the program with a run-time error.

See Also External Time Base (Synchronizing PMAC to External Events)
I-variables I12, I13
Program commands **DWELL**, **DELAY**

I12 Jog-to-Position Calculation Time

Range 1 .. 8,388,607

Units msec

Default 10

Remarks I12 controls how much time is allotted to calculate an on-line jog or homing-search move or a motion program **RAPID** move, including the "post-trigger" portions of triggered moves (homing search, move until trigger). If a motor is currently moving, it will continue on its present course during that time. If it is currently sitting still, it will continue to sit for this time.

This parameter should rarely need to be changed from the default. It should not be set to 0 for any reason, or PMAC will not be able to perform any of these types of moves. The minimum practical value for this parameter is 2 or 3.

See Also I-variables I11, I13
Program command **RAPID**
On-line commands J=, J={constant}, J/, J^{constant}, J:{constant}

I13 Programmed Move Segmentation Time

Range 0 .. 8,388,607

Units msec

Default 0

Remarks I13 controls how PMAC performs its interpolation calculations for **LINEAR** and **CIRCLE** mode moves. If I13 is set to 0, PMAC interpolates directly from the starting point of the programmed move to the ending point. This mode creates satisfactory linear interpolation in Cartesian systems, but cannot generate circular paths.

When I13 is set greater than 0, this puts PMAC into a mode ("segmentation mode") where all **LINEAR** and **CIRCLE** moves are done as a continuous cubic spline in which the move segments are of the time length specified by the parameter in this variable (this is not the same thing as **SPLINE** mode moves). This mode is required for applications using **CIRCLE** mode moves.

Segmentation mode (I13 greater than 0) is required to support any of the following PMAC features:

- Circular interpolation
- Cutter radius compensation
- **/** Program stop command
- **** Program hold command
- Rotary buffer blend on-the-fly
- Special multiple-move lookahead (Option 6L firmware)

If none of these features is required, it is usually best to leave I13 at 0, for more efficient computation.

Typical values of I13 for segmentation mode are 5 to 10 msec. The smaller the value, the tighter the fit to the true curve, but the more computation is required for the moves, and the less is available for background tasks. If I13 is set too low, PMAC will not be able to do all of its move calculations in the time allotted, and it will stop the motion program with a run-time error.

> ***Note:***
>
> When I13=0, moves are done without this ongoing spline technique, and **CIRCLE** mode moves are done as **LINEAR** mode moves.

See Also Circular Interpolation, Cutter Radius Compensation (Writing a Motion Program)
On-line commands **/**, ****
Program commands **{axis}{data}{vector}{data}, CIRCLE1**, **CIRCLE2, CC0, CC1, CC2**

I14 Auto Position Match on Run Enable

Range 0 .. 1

Units none

Default 1

Remarks I14, when set to 1, performs an automatic re-matching of motor and axis *starting* position registers to current motor *commanded* positions whenever a motion program is started. A mismatch can occur whenever a motor move (jog, open-loop, abort, or limit) changes the motor's target position without letting the axis position "know" of the change, or on power-up when an absolute position sensor starts up with a position other than zero.

With I14=1, PMAC will execute the **PMATCH** function on any Run or Step command to make sure that the axes in the motion program have the proper starting-position information. The only users who would not want this function, setting I14 to 0, are those who cannot afford the extra millisecond (approximately) of calculation time this requires.

With I14=0, PMAC uses the last motion program target position as the starting point for the calculations of the next move, even if these do not match the positions currently commanded for the motors assigned to the axes.

See Also Axis-Motor Position Re-Matching (Setting Up a Coordinate System)
On-line command **PMATCH**
Suggested M-variable definitions Mx61, Mx63, Mx64 Mx65

I15 Degree/Radian Control for User Trig Functions

Range 0 .. 1

Units none

Default 0 (degrees)

Remarks I15 controls whether the angle values for trigonometric functions in user programs (motion and PLC) and on-line commands are expressed in degrees (I15=0) or radians (I15=1).

See Also **SIN**, **COS**, **TAN**, **ASIN**, **ACOS**, **ATAN**, **ATAN2** (Computational Features)

I16 Rotary Buffer Request On Point

Range 0 .. 8,388,607

Units Command lines.

Default 5

Remarks I16 controls the point at which an executing rotary program will signal that it is ready to take more command lines (BREQ line taken high, coordinate system Rotary Buffer Full status bit – Y:$0817 bit 11 – taken low).

This occurs when the executing point in the program has caught up to within fewer lines behind the last line sent to PMAC than the value in this parameter. This can be detected as an interrupt to the host or be checked by the host on a polled basis.

Note:

The BREQ line to the interrupt controller reflects the status of the hardware-selected coordinate system (by JPAN pins FPDn/) if the control-panel inputs are enabled (I2=0); it represents the status of the software-host-addressed coordinate system if the control-panel inputs are disabled (I2=1). In virtually all applications using this feature, the user will want to set I2 to 1 so the BREQ line reflects the status of the coordinate system to which he is currently talking.

Example With I17=10 and I16=5, as program lines are sent to PMAC, PMAC will keep requesting more lines (BREQ goes high, Rotary Buffer Full bit stays 0) until there are 10 lines in the buffer ahead of the executing line. BREQ will then be held low and Rotary Buffer Full bit stays 1 until enough program lines have executed so that there are less than 5 lines in the buffer ahead of the execution point. At this time, BREQ will be set high again, and Rotary Buffer Full will become 0.

See Also Using Interrupts (Writing a Host Communications Program)
Rotary Motion Program Buffers (Writing a Motion Program)
Coordinate-system Rotary Buffer Full status bit (Y:$0817, etc., bit 16)
On-line commands **PR**, **??**
I-variables I2, I17, I18

I17 Rotary Buffer Request Off Point

Range 0 .. 8,388,607

Units Program lines

Default 10

Remarks I17 controls how many lines ahead of the executing line the host can provide a PMAC rotary motion program buffer before it signals that it is not ready for more lines (BREQ line held low, coordinate system status bit Rotary Buffer Full becomes 1). This status information can be detected either by polling (**??**or **PR**) or by using the interrupt line to the host.

If you send a program line to the rotary buffer, the BREQ line will be taken low (at least momentarily). If there are still fewer than I17 number of lines in the buffer ahead of the executing line, the BREQ line will be taken high again (giving the ability to generate an interrupt), and the Rotary Buffer Full status bit will stay 0. If there are greater than or equal to I17 lines in the buffer ahead of the executing line, the BREQ line will be left low, and the Rotary Buffer Full status bit will become 1.

Normally at this point, the host will stop sending program lines (although this is not required) and wait for program execution to catch up to within I16 lines and take BREQ high again.

Note:

The BREQ line to the interrupt controller reflects the status of the hardware-selected coordinate system (by JPAN pins FPDn/) if the control-panel inputs are enabled (I2=0); it represents the status of the software-host-addressed coordinate system if the control-panel inputs are disabled (I2=1). In virtually all applications using this feature, the user will want to set I2 to 1 so the BREQ line reflects the status of the coordinate system to which he is currently talking.

See Also Program Using Interrupts (Writing a Host Communications Program)
Rotary Motion Program Buffers (Writing a Motion Program)
Coordinate-system "buffer-full" status bit (Y:$0817, etc., bit 16)
On-line commands **PR**, **??**
I-variables I2, I16, I18

I18 Fixed Buffer Full Warning Point

Range 0 .. 8,388,607

Units Long Memory Words

Default 10

Remarks I18 sets the level of open memory below which BREQ (Buffer Request) will not go true (global status bit Fixed Buffer Full will become 0) during the entry of a fixed (non-rotary) buffer.

Every time a command line is downloaded to an open fixed buffer (PROG or PLC), the BREQ line will be taken low (at least momentarily). If there are more than I18 words of open memory left, the BREQ line will be taken high again (giving the ability to generate an interrupt), and Fixed Buffer Full will stay at 0. If there are I18 words or less, the BREQ line will be left low, and Fixed Buffer Full will become 1.

The number of available words of memory can be found using the **SIZE** command.

See Also Using Interrupts (Writing a Host Communications Program)
Global Fixed Buffer Full status bit (Y:$0003 bit 11),
I-variables I16, I17.
On-line command **SIZE**

Data Gathering I-Variables

I19 Data Gathering Period (in Servo Cycles)

Range 0 .. 8,388,607

Units Servo Interrupt Cycles

Default 0

Remarks I19 controls how often data gathering is performed, in numbers of servo interrupt cycles. If I19 is 0, data gathering is performed only once per command.

> ***Note:***
>
> Normally this parameter is controlled automatically by the PMAC Executive Program's Gathering and Tuning routines.

See Also Data Gathering (Analysis Features)
I-variables; I20-I44.
On-line commands **GATHER**, **ENDGATHER**

I20 Data Gathering Selection Mask

Range $000000 .. $FFFFFF (0 .. 16,777,215)

Units none

Default 0

Remarks I20 is a 24-bit variable that controls which of the 24 potential data sources (as specified by I21 to I44) will be gathered when gathering is performed. If bit 0 (least significant bit) is 1, the 1st source (specified by I21) will be gathered; if it is 0, it will not be. Bit 1 controls the 2nd source (I22), and so on, to bit 23, which controls the 24th source (I44).

With I9 at 2 or 3, the value of this variable will be reported back to the host in hexadecimal form, which is the more convenient form for understanding the value.

Note:

Normally this parameter is controlled automatically by the PMAC Executive Program's Gathering and Tuning routines.

Example With I20=7, only the addresses specified by I21, I22, & I23 will be gathered
With I20=$FF (255), the addresses specified by I21-I28 will be gathered
With I20=$300 (768), the addresses specified by I29 and I30 will be gathered
With I20=$FFFFFF (8,388,607), the addresses specified by I21-I44 will be gathered

See Also Data Gathering (Analysis Features)
On-line commands **GATHER**, **ENDGATHER**, **<CTRL-E>**
I-variables; I19, I21-I44.

I21 Data Gathering Source 1 Address

Range $000000 .. $FFFFFF (0 .. 16,777,215)

Units Modified PMAC addresses

Default 0

Remarks I21 specifies the address of the first data item to be gathered. This address is usually given in hexadecimal (i.e. preceded by a '$').

The specification is twenty-four bits – six hex digits. The lowest 16 bits – 4 hex digits – represent the actual word address in PMAC's memory and I/O space.

The highest two bits specify which part of the double word at that address is to be gathered (the Motorola DSP56000 has a double memory space – X and Y – to supply its dual data buses). If both of these bits are 0 – first hex digit is 0 – the Y word will be gathered; if the higher bit is 0 and the second bit is 1 – first hex digit is 4 – the X word will be gathered; if the higher bit is 1 – first hex digit is 8 or greater – both the X and Y words will be gathered as a long (double) word. In the case of a long word, it does not matter to PMAC what the second bit is, but the PMAC PC-Executive Program uses this bit to note whether this word is a fixed-point (0) or floating-point (1) value.

With I9 at 2 or 3, the value of this variable will be reported back to the host in hexadecimal form, which is the more convenient form for understanding the value.

Note:

Normally this parameter is controlled automatically by the PMAC Executive Program's Gathering and Tuning routines.

Example If the word address were $0720 (1824 decimal), I21=$000720 would denote gathering of the Y word; I21=$400720 would denote gathering of the X word; I21=$800720 or I21=$C00720 would cause gathering of both words. (You may specify this parameter in decimal form, but it is much more tricky.)

See Also Data Gathering (Analysis Features)
On-line commands **GATHER**, **ENDGATHER**, **<CTRL-E>**
I-variables I19, I21-I44.

I22–I44 Data Gathering Source 2 thru 24 Addresses

Range $000000 .. $FFFFFF (0 .. 16,777,215)

Units Modified PMAC addresses

Default 0

Remarks I22 – I44 control the addresses of the second thru twenty-fourth data items to be gathered. See I21 for more details.

> ***Note:***
>
> Normally these parameters are controlled automatically by the PMAC Executive Program's Gathering and Tuning routines.

See Also Data Gathering (Analysis Features)
On-line commands **GATHER**, **ENDGATHER**, **<CTRL-E>**
I-variables I19, I21-I44.

I45 Data Gathering Buffer Location and Mode

Range 0 .. 3

Units none

Default 0

Remarks I45 controls where the data gathering buffer will be located when it is defined, and whether it will wrap around when it is filled. It can take the following values:

0: .....Locate buffer in regular RAM. Do not permit wrap-around (stop gathering when end of buffer is reached).

1: .....Locate buffer in regular RAM. Permit wraparound upon .....reaching end of buffer. Note: Wraparound feature not .....supported by PMAC Executive program data gathering and .....tuning routines.

2: .....Locate buffer in dual-ported RAM (PMAC Option 2 .....required). Do not permit wraparound. Not very useful.

3.....Locate buffer in dual-ported RAM (PMAC Option 2 .....required). Permit wraparound upon reaching end of .....buffer (usual mode for dual-ported RAM).

When I45 is set to 2 or 3, the gather buffer starts at PMAC address $D240 – host address [base + $0900] – and occupies the number of PMAC addresses specified in the **DEFINE GATHER** command.

The DPRAM locations used by PMAC for gathering are as follows;

Address	Description
0x08FC (Y:$D23F)	Data Gather Buffer Size.
0x08FC (X:$D23F)	PMAC Data Gather Buffer Storage Address. If I45 = 2 and the buffer's end has been reached (this index is greater than or equal to the size), the **DEFINE GATHER** command must be issued again to allow gathering to restart.
0x0900 ($D240)	Start of Data Gather Buffer (not changeable).

Note:

In firmware version 1.16B and older, these addresses were 0x0100 ($0040) lower.

See Also Data Gathering (Analysis Features)
Option 2 Dual-Ported RAM
On-line commands **DEFINE GATHER**, **GATHER**, **ENDGATHER**, **DELETE GATHER**, **LIST GATHER**

I46 CPU Frequency Control {PMAC w/Flex CPU only}

Range 0 .. 15

Units Multiplication Factor

Default 0 (jumper-set frequency)

Remarks I46 can control the operational clock frequency of the CPU in an Option 5xF "Flex" CPU by controlling the multiplication factor of the phase-locked loop (PLL) inside the CPU. The PLL circuit multiplies the input 10 MHz (actually 9.83 MHz) clock frequency by a factor of (I46 + 1) to create the clock frequency for the CPU. Formally, this is expressed in the equation:

*CPU Frequency (MHz) = 10 * (I46 + 1)*

If I46 is set to 0, or an older style of CPU (not "Flex") is used, the CPU frequency is set by jumpers (E48 on a PMAC(1); E2 and E4 on a PMAC2).

I46 should usually be set to create the highest CPU frequency for which the CPU is rated. For the Option 5AF 40 MHz CPU, it should be set to 3; for the Option 5CF 80 MHz CPU, it should be set to 7; for the Option 5EF 160 MHz CPU, it should be set to 15. With any of the Flex CPU options (5xF), the PMAC will not permit the CPU to run at higher than the rated frequency, and it will reduce I46 to the matching value..

I46 is actually used at power-on/reset only, so to make a change in the CPU frequency with I46, change the value of I46, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command.

I47 Address of Pointer for Control-W Command

Range $0000 .. $FFFF (0 .. 65,535)

Units Legal PMAC 'Y' addresses

Default 0

Remarks I47 specifies the address of the register that tells the **<CONTROL-W>** command where to pick up its command string.

The **<CONTROL-W>** command permits the host to load command strings into dual-ported RAM (Option 2 required), instead of the normal command interface, then cause the command to be accepted by sending a single byte (ASCII 23D is **<CTRL-W>**) to the command interface.

> ***Note:***
>
> The **<CONTROL-W>** function is now effectively obsolete. The newer bidirectional DPRAM ASCII communications feature enabled by I58 is superior and should be used instead.

Example For instance, if I47 is set to $D200, PMAC will look to its memory register Y:$D200 (wherever it sits in the host memory space) on receipt of a **<CTRL-W>** to see where to look for the command string. If Y:$D200 holds a value of $D700, PMAC will take the command string starting at register Y:$D700, incrementing addresses until it finds the null character (value 0).

See Also Option 2 Dual-Ported RAM Manual
Memory-map registers $D000-$DFFF.
I-variables I56, I58
On-line command **<CONTROL-W>**

I48 DPRAM Servo Data Enable

Range 0 .. 1

Units none

Default 0

Remarks I48 enables or disables the dual-ported RAM (DPRAM) servo data reporting function. When I48=1 and the **GATHER** command has been issued, PMAC copies key data from the servo control registers to fixed registers in the DPRAM every I19 servo cycles for easy access by the host computer. Servo data for motors up to the number specified by I59 are reported.

When I48=0, the DPRAM servo data reporting function is disabled. Regular data gathering can be enabled in this mode.

Refer to the description of DPRAM functions for more information.

See Also DPRAM Servo Data Reporting (Option 2 DPRAM Manual)
I-variables I19, I49, I59
On-line commands **GATHER**, **ENDGATHER**

I49 DPRAM Background Data Enable

Range 0 .. 1

Units none

Default 0

Remarks I49 enables or disables the dual-ported RAM (DPRAM) background data reporting function. When I49=1, PMAC copies key data from the background information registers to fixed registers in the DPRAM for easy access by the host computer. Each time the host computer reads these registers and signals it is done, PMAC will copy the data again. Data for motors and coordinate systems up to the number specified by I59 are reported.

When I49=0, the DPRAM background data reporting function is disabled.

Refer to the description of DPRAM functions for more information.

See Also DPRAM Background Data Reporting (Option 2 DPRAM Manual)
I-variables I48, I59

I50 Rapid Move Mode Control

Range 0 .. 1

Units none

Default 1

Remarks I50 determines which variables are used for speed of **RAPID** mode moves. When I50 is set to 0, the jog parameter for each motor (Ix22) is used. When I50 is set to 1, the maximum velocity parameter for each motor (Ix16) is used instead. Regardless of the setting of I50, the jog acceleration parameters Ix19-Ix21 control the acceleration.

See Also **RAPID** mode moves (Writing a Motion Program)
I-variables Ix16, Ix19-Ix22
Program command **RAPID**.

I51 Compensation Table Enable

Range 0 .. 1

Units none

Default 0

Remarks I51 permits the enabling and disabling of the PMAC's compensationtables – leadscrew (position) compensation tables, torque compensation tables, and backlash compensation tables. When I51 is 0, all tables are disabled and there is no correction performed. When I51 is 1, all existing tables are enabled and corrections are performed as specified in the tables.

See Also Leadscrew Compensation (Setting Up a Motor)
On-line commands **DEFINE COMP**, **DELETE COMP**, **LIST COMP DEF**.
Position-compensation registers (D:$46, etc.)
Torque compensation registers (Y:$45, etc.)
Suggested M-variables Mx69

I52 \ Program Hold Slew Rate

Range 0 .. 8,388,607

Units I10 units / segmentation period

Default 37,137

Remarks I52 controls the slew rate to a stop on a **** program hold command, and the slew rate back up to speed on a subsequent **R** command, for all coordinate systems, provided PMAC is in a segmented move (**LINEAR** or **CIRCLE** mode with I13>0). If PMAC is not in a segmented move (I13=0, or other move mode), the **** command acts just like an **H** feed hold command, with Ix95 controlling the slew rate.

The units of I52 are the units of I10 (1/8,388,608 msec) per segmentation period (I13 msec). To calculate how long it takes to stop on a **** command, and to restart on the next R command, use the formula

*T (msec) = I10 * I13 / I52*

To calculate the value of I52 for a given start/stop time, use the formula

*I52 = I10 * I13 / T (msec).*

Example To execute a full stop in one second with the default servo update time (I10 = 3,713,707) and a move segmentation time of 10 msec, I52 should set to 3,713,707 * 10 / 1000 = 37,137.

See Also Stop Commands (*Making Your Application Safe*)
I-variables I13, Ix95
On-line commands ****, **H**

I53 Program Step Mode Control

Range 0 .. 1

Units none

Default 0

Remarks I53 controls the action of a Step (**S**) command in any coordinate system on PMAC. At the default I53 value of zero, a Step command causes program execution through the next move, **DELAY**, or **DWELL** command in the program, even if this takes multiple program lines.

When I53 is set to 1, a Step command causes program execution of only a single program line, even if there is no move or **DWELL** command on that line. If there is more than one **DWELL** or **DELAY** command on a program line, a single Step command will only execute one of the **DWELL** or **DELAY** commands.

Regardless of the setting of I53, if program execution on a Step command encounters a **BLOCKSTART** statement in the program, execution will continue until a **BLOCKSTOP** statement is encountered.

See Also Control Panel Port STEP/ Input (Connecting PMAC to the Machine)
On-line commands **<CTRL-R>**, **<CTRL-S>**, **Q**, **R**, **S**
Program commands **BLOCKSTART**, **BLOCKSTOP**

I54 Serial Baud Rate {PMAC(1) w/Flex CPU or PMAC2 only}

Range 0 .. 15

Units none

Default 8 (9600 baud) PMAC(1)
12 (38400 baud) PMAC2

Remarks I54 controls the baud rate for communications on the serial port for all PMAC2 boards, and for PMAC(1) boards with an ACC-5xF Flex CPU. PMAC2 uses I54 only at power-up/reset to set up the frequency of the clocking circuit for the serial port.

To change the baud rate, it is necessary to change the value of I54, store this value to non-volatile flash memory with the **SAVE** command, and reset the card. At this time, PMAC2 will establish the new baud rate.

The possible settings of I54 and the baud rates they define are:

I54	Baud Rate	Error with CPU at 40 MHz	Error with CPU at 60 MHz	Error with CPU at 80 MHz	Error with CPU at 160 MHz
0	600	0	(Disabled)	0	0
1	900	-0.05%	0	-0.03%	-0.01%
2	1200	0	0	0	0
3	1800	-0.1%	0	-0.05%	-0.03%
4	2400	0	0	0	0
5	3600	-0.19%	0	-0.10%	-0.05%
6	4800	0	0	0	0
7	7200	-0.38%	0	-0.19%	-0.10%
8	9600	0	0	0	0
9	14,400	-0.75%	0	-0.38%	-0.19%
10	19,200	0	0	0	0
11	28,800	-1.5%	0	-0.75%	-0.38%
12	38,400	0	0	0	0
13	57,600	-3.0%	0	-1.5%	-0.75%
14	76,800	0	0	0	0
15	115,200	(Disabled)	0	-3.0%	-1.5%

CPUs run at 30 MHz, 90 MHz, 120 MHz, or 150 MHz, as well as 60 MHz, also have zero baud rate errors at all of these baud rates. Some users may want to slow down their CPU frequencies from the maximum rated frequency in order to get accurate high baud rates.

Because of the nature of the clock generation circuitry, odd values of I54 on a PMAC2 with a CPU operating at 40 MHz (Jumper E2 OFF) produce non-exact baud rates. The error in baud rate is small enough that communications should still be valid.

If your host computer baud rate cannot be made to match the PMAC2's baud rate, either PMAC2's baud rate must be changed through the bus communications port, or the PMAC2 must be re-initialized by resetting or powering up with the E3 jumper ON. This forces the PMAC2 to the default baud rate of 38,400.

I55 DPRAM Background Variable Buffers Enable

Range 0 .. 1

Units none

Default 0

Remarks I55 enables or disables the dual-ported RAM (DPRAM) background variable read and write buffer function. When I55 is 0, this function is disabled. When I55 is 1, this function is enabled. When enabled, the user can specify up to 128 PMAC registers to be copied into DPRAM each background cycle to be read by the host (background variable read) and up to 128 PMAC registers to be copied each background cycle from values written into the DPRAM by the host (background variable write).

See Also DPRAM Background Variable Read Buffer (Option 2 DPRAM Manual)
DPRAM Background Variable Write Buffer (Option 2 DPRAM Manual)

I56 DPRAM ASCII Communications Interrupt Enable

Range 0 .. 1

Units none

Default 0

Remarks I56 enables or disables the interrupt feature for the dual-ported RAM (DPRAM) ASCII communications function that is enabled with I58=1. When I56=1, PMAC will generate an interrupt to the host computer each time it loads a line into the DPRAM ASCII buffer for the host to read. When I56=0, it will not generate this interrupt.

For PMAC(1)-PC, PMAC(1)-Lite, PMAC(1)-PCI, and PMAC(1)-PCI-Lite, the interrupt line used is the EQU4 interrupt. For this to reach the host, the E55 jumper must be ON, and E54, E56, and E57 must be OFF (E54 and E56 do not exist on PMAC-Lite). When using this feature, do not use the EQU4 line for any other purpose, including position compare.

For the regular (non-Ultralite) ISA-bus and PCI-bus versions of PMAC2 (PMAC2-PC, PMAC2-Lite, Mini-PMAC2, PMAC2-PCI, PMAC2-PCI-Lite), the interrupt line used is the EQU1 interrupt. When using this feature, do not use the EQU1 line for any other purpose, including position compare.

For the PMAC2-PC Ultralite and PMAC2-PCI Ultralite, the interrupt line used is the CTRL0 line, which has no other functions.

For the VME-bus versions (PMAC(1)-VME, PMAC2-VME, PMAC2-VME Ultralite), the interrupt line used is the normal communications interrupt (the only interrupt available). This line – IRQn on the VME bus, is determined by the VME setup value in register X:$0788. The interrupt vector provided to the host is one greater than the value in setup register X:$0789. For example, if the value in X:$0789 is the default of $A1, this interrupt will provide an interrupt vector of $A2.

See Also DPRAM ASCII Communications (Option 2 DPRAM Manual)
I-variables I48, I49, I55, I57, I58, I59

I57 DPRAM Binary Rotary Buffer Enable

Range 0 .. 1

Units none

Default 0

Remarks I57 enables or disables the dual-ported RAM (DPRAM) binary rotary buffer function. When I57=1, this function is enabled and the host computer can download motion program data to the PMAC through the DPRAM in binary form for maximum possible throughput. When I57=0, this function is disabled.

See Also DPRAM Binary Rotary Buffer (Binary Rotary Buffer)
I-variables I48, I49, I55, I56, I58, I59

I58 DPRAM ASCII Communications Enable

Range 0 .. 1

Units none

Default 0

Remarks I58 enables or disables the dual-ported RAM (DPRAM) ASCII communications function. When I58=1, this function is enabled and the host computer can send ASCII command lines to the PMAC through the DPRAM and receive ASCII responses from PMAC through the DPRAM. When I58=0, this function is disabled.

With I58=1, PMAC's response port on the bus will be whichever port (normal I/O-mapped bus port, or DPRAM) has received the most recent command.

The **<CTRL-Z>** command, which causes PMAC's response port to revert to the serial port, automatically sets I58 to 0.

When I58 is set to 1, I3 is automatically forced to 0 or 2, if it has been at 1 or 3, respectively.

If I56 is also equal to 1, PMAC will provide an interrupt to the host computer when it provides a response string.

See Also DPRAM ASCII Communications (Option 2 DPRAM Manual)
I-variables I3, I48, I49, I55, I56, I57, I59
On-line command **<CTRL-Z>**

I59 DPRAM Buffer Maximum Motor/C.S. Number

Range 0 .. 8

Units none

Default 0

Remarks I59 determines the highest-numbered motor and/or coordinate system for which servo data is reported in the DPRAM when servo data reporting is active (I48=1) and/or background data reporting is active (I49=1). If I59=0, no data is reported even if one or both reporting functions are active. If I59>0, data for both motor and coordinate system of numbers up to the value of I59 are reported – there is not separate control of maximum motor and coordinate system numbers.

See Also DPRAM Servo Data Reporting, DPRAM Background Data Reporting

I60 Auto-Converted ADC Register Address {PMAC(1) only}

Range 0, $FFD0 .. $FFFE

Units PMAC "Y" addresses

Default 0

Remarks I60 permits the user to specify the address of one ACC-36 analog-to-digital converter (ADC) board (or on-board Option 12 ADCs on PCI-bus PMACs) whose values will automatically be copied into PMAC(1)'s memory at a high rate so that they can be used as servo feedback. This can also be used to make user program access to these ADCs more convenient, but it is not required for this purpose.

On a PMAC2 board, this function is controlled by the more flexible structure of the analog data table.

There are 24 legal addresses for an ACC-36 in PMAC's memory and I/O space: even values from $FFD0 to $FFFE. The Option 12 ADCs on a PCI-bus PMAC(1) are located at address $FFC8. If you have more than one ACC-36 connected to PMAC, only one board may be used in this manner. All other boards must be accessed in user programs.

For the ACC-36 board automatically converted using I60 and I61, the board must never be accessed in user programs, but user programs may read the memory registers in PMAC to which the ADC values are copied.

If I60 is set to 0, no automatic conversion will take place. If the first two hex digits of I60 are set to anything except $FF, PMAC will automatically change them to $FF.

ADCs 1 to 8 are copied into the low 12 bits of registers Y:$0708 to Y:$070F, respectively. ADCs 9 to 16, if they exist on the addressed board, are copied into the low 12 bits of registers X:$0708 to X:$070F. These registers should be treated as signed registers.

Note:

It is easier to specify this parameter in hexadecimal form ($ prefix). If I9 is set to 2 or 3, the value of this variable will be reported back to the host in hexadecimal form.

Example A PMAC system has an ACC-14D at address $FFD0, and an ACC-36 at address $FFD8. It is desired to automatically convert all 8 registers on the ACC-36. I60 is set to $FFD8, and I61 is set to 7.

See Also Parallel Position Feedback Conversion (Setting Up a Motor)
I-variables I61, Ix10
Memory and I/O Map registers $FFD0 to $FFFE
ACC-36 User's Manual

I61 Number of Auto-Converted ADC Registers {PMAC(1) only}

Range 0 .. 7

Units Number of registers minus 1

Default 0

Remarks I61 permits the user to specify the number of analog-to-digital converter (ADC) registers on the ACC-36 specified by I60 that will be automatically converted and copied into PMAC(1) memory. There are two 12-bit converters per 24-bit register. The number of registers converted automatically is equal to I61 + 1.

On a PMAC2 board, this function is controlled by the more flexible structure of the analog data table.

Each phase cycle (9 kHz default), PMAC copies the contents of an ACC-36 register into RAM, then selects the next register, so the conversion can start and the results will be ready for the next phase cycle. PMAC will cycle through the first I61+1 registers on the ACC-36 in this fashion. If I61 is set to 0, PMAC will cycle through all 8 registers on the ACC-36 (equivalent to I61=7).

If you have more than one ACC-36 connected to PMAC, only one board may be used in this manner. All other boards must be accessed in user programs. For the ACC-36 board automatically converted using I60 and I61, the board must never be accessed in user programs, but user programs may read the memory registers in PMAC to which the ADC values are copied.

ADCs 1 to 8 are copied into the low 12 bits of registers Y:$0708 to Y:$070F, respectively. ADCs 9 to 16 are copied into the low 12 bits of registers X:$0708 to X:$070F. These registers should be treated as signed values.

Example The system has 8 axes with analog feedback. There are 4 phase cycles per servo cycle,

and it is important to have new feedback values every servo cycle. Therefore an ACC-36 with Option 1 is ordered, so there are 2 ADCs per register, and I61 is set to 3 to convert the first 4 registers in a cyclic fashion. ADCs 1 to 4 are copied into Y:$0708 to Y:$070B, respectively; ADCs 9 to 12 are copied into X:$0708 to X:$070B, respectively.

See Also Parallel Position Feedback Conversion (Setting Up a Motor)
I-variables I60, Ix10
Memory and I/O Map registers $FFD0 to $FFFE
ACC-36 User's Manual

I62 Internal Message Carriage Return Control

Range 0 .. 1

Units none

Default 0

Remarks I62 permits the user to control whether internally generated messages sent from PMAC to the host computer are terminated with the carriage return (**<CR>**) character or not. It affects only those messages generated by a **CMD**, **SEND**, **SENDP**, or **SENDS** statement in a PMAC motion or PLC program. The ability to suppress the **<CR>** provides more flexibility in controlling the format display of a terminal window or printer.

If I62 is set to the default value of 0, these messages are terminated with a **<CR>**. If I62 is set to 1, the **<CR>** is suppressed. With I62 set to 1, if it desired for a PMAC program to cause a **<CR>** to be sent, the **SEND^M** command must be used (the carriage return character is **<CTRL-M>**).

> ***Note:***
>
> Do not set I62 to1 if using dual-ported RAM ASCII communications (I58=1).

Example With program code:

```
I62=1.....                    ; Suppress <CR> on SEND
SEND "THE VALUE OF P1 IS "    ; String sent with no <CR>
CMD "P1" .....                ; Response string follows on same line, no <CR>
SEND^M.....                   ; Send a <CR>
```

PMAC responds with:

```
THE VALUE OF P1 IS 42
```

See Also Program Commands **CMD, SEND, SENDS, SENDP**

I63 Control-X Echo Enable

Range 0 .. 1

Units None

Default 0

Remarks I63 permits the PMAC to echo the **<CONTROL-X>** character back to the host computer when it is received. If I63 is set to 1, PMAC will send a **<CONTROL-X>** character (ASCII value 24 decimal) back to the host computer when it receives a **<CONTROL-X>** character.

If I63 is set to 0, PMAC will send nothing back to the host computer when it receives a **<CONTROL-X>** character. This is equivalent to the action of older versions of PMAC firmware without an I63 variable.

The host computer can use the **<CONTROL-X>** character to clear out PMAC's communications buffers and make sure that no unintended responses are received for the next command. However, without an acknowledgement that the buffers have been cleared, the host computer has to add a safe delay to ensure that the operation has been done before the next command can be issued.

Setting I63 to 1 permits a more efficient clearing of the buffer, because the response character lets the host computer know when the next command can safely be sent.

Versions of the PCOMM32 communications library 2.21 and higher (March 1999 and newer) can take advantage of this feature for more efficient communications. I63 should be set to 0 when using older versions of PCOMM32.

In battery-backed PMAC(1) boards with firmware versions 1.16F and 1.16G, the value of I63 is maintained by the battery through a power cycling or reset; a **SAVE** command is not required. In 1.16H and newer (and in all revisions on flash-backed boards), the value is maintained by storing it to non-volatile memory with a **SAVE** command.

I64 Internal Response Tag Enable

Range 0 .. 1

Units None

Default 0

Remarks I64 permits PMAC to tag ASCII text lines that it sends to the host computer as a result of internal commands, so these can easily be distinguished from responses to host commands.

If I64 is set to 1, a line of text sent to the host computer as a result of an internal **SEND** or **CMD** statement is preceded by a **<CONTROL-B>** ("start-transmission") character. In the case of an error report, the **<CONTROL-B>** character replaces the leading **<CONTROL-G>** ("bell") character. The text line is always terminated by a **<CR>** (carriage return) character, regardless of the setting of I62.

If I64 is set to 0, a text line sent in response to an internal PMAC command is not preceded by any special character. Reported errors are preceded by the **<CONTROL-G>** ("bell") character. This is equivalent to the action of older versions of PMAC firmware, before I64 was implemented.

Regardless of the setting of I64, if I6 = 2, errors on internal commands are not reported to the host computer.

In battery-backed PMAC(1) boards with firmware versions 1.16F and 1.16G, the value of I64 is maintained by the battery through a power cycling or reset; a **SAVE** command is not required. In 1.16H and newer (and in all revisions on flash-backed boards , the value is maintained by storing it to non-volatile memory with a **SAVE** command.

Example With I64=0, lines sent from PMAC are:

```
Motion Stopped on Limit<CR>
<BELL>ERR003<CR>
```

With I64=1, the same lines from PMAC are:

```
<CTRL-B>Motion Stopped on Limit<CR>
<CTRL-B>ERR003<CR>
```

I65 User-Configuration Variable

Range 0 – 16,777,215

Units None

Default 0

Remarks I65 is an I-variable that has no automatic use on PMAC. The purpose of this variable is to provide an easy way for the user to confirm that the application configuration has been loaded into the PMAC. Since the factory default value for I65 is 0, setting I65 to a non-zero value as part of the configuration permits an easy way to verify that the configuration file has been downloaded.

By providing many different possible non-zero values of I65, different machine configurations can be identified with I65. It is even possible for the user to utilize I65 as an electronic serial number.

I66 Servo-Channel ADC Auto-Copy Disable {PMAC2 only}

Range 0 .. 1

Units None

Default 0

Remarks I66 permits the disabling of the PMAC2 function that automatically copies the values in the 16 A/D-converter registers (A and B registers of Channels 1 – 8) of the two "DSPGATE1" Servo ICs into RAM every phase cycle. This auto-copying function was implemented because in the early revisions of the DSPGATE1 IC, the ADC registers themselves could only be read reliably during phase-interrupt tasks.

Note:

This function is not to be confused with de-multiplexing of Option 12 or ACC-36 ADCs controlled by I60 and I61 on a PMAC(1) or the analog table on a PMAC2.

Recent revisions of the DSPGATE1 IC ("B" revision and newer), installed on virtually all PMAC2 boards starting in the year 2000, double buffer these registers so that they may be read properly at any time. Therefore, this auto-copying function is not necessary in most cases on newer boards.

If I66 is set to the default value of 0, at the beginning of each phase cycle, PMAC2 copies the values found at these 16 addresses (whether physically present or not) into RAM registers at X/Y:$0710 – X/Y:$0717.

If I66 is set to 1, PMAC2 does not perform this copying function each phase cycle. The user may want to disable the copying for two reasons. First, it saves significant amounts of processor time. Second, the auto-copying process interferes with the operation of an ACC-51P board that is mapped into Channels 1 – 4 or 5 – 8. (It does not interfere with an ACC-51P board mapped into Channels 9 – 12 or 13 – 16.)

I66 is used at power-up/reset only. To enable or disable the auto-copying function, change the value of I66, issue the **SAVE** command, then reset the card. If you wish to temporarily enable or disable this function, change the internal control bit at X:$0003 bit 15.

Note:

The P2Setup PC program, when used to set up digital current loop operation for on-board servo channels (not through MACRO),

requires that the auto-copying function be enabled.

I67 Modbus TCP Buffer Start Address

Range $0 – $9FFF

Units PMAC addresses

Default 0

Remarks I67 enables the Modbus TCP interface in PMAC software and reports the starting address of the 256-word Modbus buffer in PMAC memory. To enable the Modbus TCP interface on the PMAC's Ethernet port, the following conditions must apply:

1. The Ethernet physical interface must be present
2. The Modbus TCP firmware for the Ethernet processor must be installed
3. V1.17C or newer PMAC firmware must be installed
4. A user buffer of 256 or more words must have been defined with the DEFINE UBUFFER command
5. I67 must be set to a value greater than 0.

The user can set I67 to any value greater than 0 to enable the Modbus TCP buffer. When this is done, PMAC will automatically set I67 to the address of the start of the 256-word Modbus buffer. In most PMAC configurations, this address will be $9F00, so the buffer will occupy the addresses $9F00 - $9FFF.

A **SAVE** command must be issued with I67 at a non-zero value in order for the Modbus TCP buffer to be active after subsequent power-up or reset operations.

I68 Alternate TWS Input Format

Range 0 – 1

Units None

Default 0

Remarks I68 controls how the PMAC interprets incoming data on a TWS-format M-variable read from an ACC-34 or similar serial-interface I/O board. If I68 is set to the default value of 0, PMAC expects the serial input data on the DAT0 signal line. If I68 is set to 1, PMAC expects the serial input data on the DAT7 signal line.

The DAT7 line is separated more from the output clock line on the same cable; the use of DAT7 by setting I68 to 1 and making the appropriate jumper setting on the I/O board makes it possible to use a longer cable without too much coupling interference between signals.

On the ACC-34AA, jumper E23 must be connect pins 1 and 2 to support the default setting of I68 = 0; it must connect pins 2 and 3 to support the setting of I68 = 1. On the ACC-76 and ACC-77 "P-Brain" boards, jumper E1 should be ON to support the default setting of I68 = 0; jumper E8 should be ON to support the setting of I68 = 1. Older boards of this class do not support settings of I68 = 1.

I69 Modbus TCP Software Control Panel Start Address

Range $0 – $FFFF

Units PMAC addresses

Default 0

Remarks I69 enables and specifies the address of the start of the Modbus TCP software control panel in PMAC. I69 permits a software control panel to be commanded over the Modbus TCP link, typically from a PLC, using part of the user buffer created with the **DEFINE UBUFFER** command and reserved for Modbus TCP use with I67. If I69 is set to a value greater than 0, this software control panel is enabled. Typically, I69 is set to a value 128 ($80) greater than the value of I67, so this control panel starts at an address 128 higher than the beginning of the entire Modbus TCP buffer. For example, if the beginning of the Modbus buffer were at $9F00, I69 could be set to $9F80.

The software control panel occupies 18 long words of PMAC memory. The structure functions of the Modbus panel are equivalent to those for the DPRAM software control panel, which are documented in the Memory and I/O Map chapter of the Software Reference Manual at their default addresses of $D000 - $D011.

The operation of the Modbus control panel is independent of that for the DPRAM control panel (which is controlled by I2). One, neither, or both of these control panels may be active at one time.

I70 – I77 Analog Table Setup Lines

Range $000000 - $FFFFFF

Units none

Default $0

Remarks PMAC2 firmware automatically selects and reads the channels of Option 12 and 12A A/D converters in a round-robin fashion. This function is controlled by a data table in variables I70 – I77 which operates much like the encoder conversion table. The eight I-variables (X registers) contain the channel-select information, and the eight Y-registers contain the A/D results. Each X and Y word is split into two 12-bit halves, where the lower 12 bits work with the first A/D converter set (Option 12), and the higher 12 bits work with the second A/D converter set (Option 12A).

The data table looks like this:

Setup I-Variable	I-Variable Upper 12 Bits	I-Variable Lower 12 Bits	Result Address	Y Word Upper 12 Bits	Y Word Lower 12 Bits
I70	CONFIG_W2	CONFIG_W1	Y:$0708	DATA_W2	DATA_W1
I71	CONFIG_W2	CONFIG_W1	Y:$0709	DATA_W2	DATA_W1
I72	CONFIG_W2	CONFIG_W1	Y:$070A	DATA_W2	DATA_W1
I73	CONFIG_W2	CONFIG_W1	Y:$070B	DATA_W2	DATA_W1
I74	CONFIG_W2	CONFIG_W1	Y:$070C	DATA_W2	DATA_W1
I75	CONFIG_W2	CONFIG_W1	Y:$070D	DATA_W2	DATA_W1
I76	CONFIG_W2	CONFIG_W1	Y:$070E	DATA_W2	DATA_W1
I77	CONFIG_W2	CONFIG_W1	Y:$070F	DATA_W2	DATA_W1

where:

CONFIG_W2 is the selection word for the second A/D converter set (Option 12A)

CONFIG_W1 is the selection word for the first A/D converter set (Option 12)

DATA_W2 is the matching A/D data from the second A/D converter set (Option 12A)

DATA_W1 is the matching A/D data from the first A/D converter set (Option 12)

A value of 0-7 in CONFIG_W1 tells PMAC2 to read channel ANAI00-07, respectively, as a 0 to+5V input, resulting in an unsigned value.

A value of 8-15 in CONFIG_W1 tells PMAC2 to read ANAI00-07, respectively, as a -2.5 to +2.5V input, resulting in a signed value.

A value of 0-7 in CONFIG_W2 tells PMAC2 to read channel ANAI08-15, respectively, as a 0 to+5V input, resulting in an unsigned value.

A value of 8-15 in CONFIG_W1 tells PMAC2 to read ANAI08-15, respectively, as a -2.5 to +2.5V input, resulting in a signed value.

Each phase update (9 kHz default), PMAC2 increments through one line of the table. It copies the ADC reading(s) selected in the previous cycle into RAM, then writes the next configuration words to the ADC(s). Typically, this will be used to cycle through all 8 ADCs or pairs of ADCs. To cycle through all 8 pairs of ADCs in unsigned mode, the table should look like this:

Setup I-Variable	X Word Upper 12 Bits	X Word Lower 12 Bits	Result Address	Y Word Upper 12 Bits	Y Word Lower 12 Bits
I70	0	0	Y:$0708	ANAI08	ANAI00
I71	1	1	Y:$0709	ANAI09	ANAI01
I72	2	2	Y:$070A	ANAI10	ANAI02
I73	3	3	Y:$070B	ANAI11	ANAI03
I74	4	4	Y:$070C	ANAI12	ANAI04
I75	5	5	Y:$070D	ANAI13	ANAI05
I76	6	6	Y:$070E	ANAI14	ANAI06
I77	7	7	Y:$070F	ANAI15	ANAI07

If you wanted to set up all ADCs for a unipolar (unsigned) conversion, the following commands could be issued

```
I70=$000000 ; Select ANAI00 and ANAI08 (if present) unipolar
I71=$001001 ; Select ANAI01 and ANAI09 (if present) unipolar
I72=$002002 ; Select ANAI02 and ANAI10 (if present) unipolar
I73=$003003 ; Select ANAI03 and ANAI11 (if present) unipolar
I74=$004004 ; Select ANAI04 and ANAI12 (if present) unipolar
I75=$005005 ; Select ANAI05 and ANAI13 (if present) unipolar
I76=$006006 ; Select ANAI06 and ANAI14 (if present) unipolar
I77=$007007 ; Select ANAI07 and ANAI15 (if present) unipolar
```

To set up the configuration words for bipolar analog inputs, the commands could look like this:

```
I70=$008008 ; Select ANAI00 and ANAI08 (if present) bipolar
I71=$009009 ; Select ANAI01 and ANAI09 (if present) bipolar
I72=$00A00A ; Select ANAI02 and ANAI10 (if present) bipolar
I73=$00B00B ; Select ANAI03 and ANAI08 (if present) bipolar
I74=$00C00C ; Select ANAI04 and ANAI08 (if present) bipolar
```

```
I75=$00D00D ; Select ANAI05 and ANAI08 (if present) bipolar
I76=$00E00E ; Select ANAI06 and ANAI08 (if present) bipolar
I77=$00F00F ; Select ANAI07 and ANAI08 (if present) bipolar
```

Once this setup has been made, PMAC2 will automatically cycle through the analog inputs, copying the converted digital values into RAM. These image registers can then be read as if they were the actual A/D converters. For user program use, the image registers would be accessed with M-variables. Suggested definitions for unipolar (unsigned) values are:

```
M1000->Y:$0708,0,12,U   ; ANAI00 image register; from J1 pin 1
M1001->Y:$0709,0,12,U   ; ANAI01 image register; from J1 pin 2
M1002->Y:$070A,0,12,U   ; ANAI02 image register; from J1 pin 3
M1003->Y:$070B,0,12,U   ; ANAI03 image register; from J1 pin 4
M1004->Y:$070C,0,12,U   ; ANAI04 image register; from J1 pin 5
M1005->Y:$070D,0,12,U   ; ANAI05 image register; from J1 pin 6
M1006->Y:$070E,0,12,U   ; ANAI06 image register; from J1 pin 7
M1007->Y:$070F,0,12,U   ; ANAI07 image register; from J1 pin 8
M1008->Y:$0708,12,12,U  ; ANAI08 image register; from J1 pin 9
M1009->Y:$0709,12,12,U  ; ANAI09 image register; from J1 pin 10
M1010->Y:$070A,12,12,U  ; ANAI10 image register; from J1 pin 11
M1011->Y:$070B,12,12,U  ; ANAI11 image register; from J1 pin 12
M1012->Y:$070C,12,12,U  ; ANAI12 image register; from J1 pin 13
M1013->Y:$070D,12,12,U  ; ANAI13 image register; from J1 pin 14
M1014->Y:$070E,12,12,U  ; ANAI14 image register; from J1 pin 15
M1015->Y:$070F,12,12,U  ; ANAI15 image register; from J1 pin 16
```

For bipolar (signed), just change the **U** in each definition to **S**.

In firmware versions prior to V1.17C, the 8 setup registers did not have I-variables assigned to them, but operated in the same way, accessed directly at addresses X:$0708 – X:$070F.

I8x Motor x Third-Resolver Gear Ratio

Range 0 .. 4095

Units Second-resolver turns per third-resolver turn

Default 0

Remarks I8x tells PMAC the gear ratio between the second (medium) and third (coarse) resolvers for a triple-resolver setup for Motor x. It is expressed as the number of turns (electrical cycles) the second resolver makes in one full turn (electrical cycle) of the third resolver.

This parameter is used only during PMAC's power-up/reset cycle to establish absolute power-on servo position. Therefore, the parameter must be set, the value stored in EAROM with the **SAVE** command, and the card reset before it takes effect.

If there is no geared third resolver on Motor x, or if absolute power-on position is not desired, I8x should be set to zero. If either Ix10 (for the primary resolver) or I9x (for the secondary resolver) is set to zero, I8x is not used.

The third resolver must be connected to the next higher numbered R/D converter at the same multiplexer address than the second resolver, which must be connected to the next higher numbered converter at the same multiplexer address than the primary resolver.

There can be up to eight R/D converters on two ACC-8D Option 7 boards at one multiplexer address.

Example Motor 3 has a triple resolver, with each resolver geared down by a ratio of 16:1 from the resolver before it. The fine resolver is connected to R/D converter 4 at multiplexer address 0 (the first R/D converter on the second ACC-8D Option 7 at address 0). The medium resolver is connected to R/D converter 5 at this address, and the coarse resolver is connected to R/D converter 6. The following I-variable values should be used:

```
I310=$040100 ....     ; The 0100 in the low 16 bits specifies
.....                 ; multiplexer address 0; the 4 in the high 8 bits
.....                 ; specifies R/D converter 4 at this address.
I93=16 .....          ; Specifies 16:1 ratio between medium and fine
I83=16                ; Specifies 16:1 ratio between coarse and medium
```

See Also Selecting the Position Loop Feedback (Setting Up a Motor)
I-Variables I9x, Ix03, Ix10, Ix81
ACC-8D Option 7 (R/D Converter) Manual

I89 Cutter Comp Outside Corner Break Point

Range -1.0 – 0.99999

Units cos Δθ

Default 0.99848 (cos 1^{o})

Remarks I89 controls the threshold between outside corner angles for which an extra arc move is added in cutter compensation, and those for which the incoming and outgoing moves are directly blended together.

I89 is expressed as the cosine of the change in directed angle between the incoming and outgoing moves. As such, it can take a value between -1.0 and +1.0. If the two moves have the same directed angle at the move boundary (i.e. they are moving in the same direction), the change in directed angle is 0, and the cosine is 1.0. As the change in directed angle increases, the corner gets sharper, and the cosine of the change in directed angle decreases. For a total reversal, the change in directed angle is 180^{o}, and the cosine is -1.0.

If the cosine of the change in directed angle of an outside corner is less than I89 (a large change in directed angle; a sharp corner), PMAC will automatically add an arc move with a radius equal to the cutter radius to join the incoming and outgoing moves. This prevents the cutter from moving too far out when going around the outside of a sharp corner.

If the cosine of the change in directed angle of an outside corner is greater than I89 (a small change in directed angle; a gradual corner), PMAC will directly blend the incoming and outgoing moves with its normal blending algorithms. This can provide increased speed on small angle changes, because an extra segment of minimum TA or 2*TS time is not added.

Note:

Do not set I89 to +1.0 (or greater). Otherwise, PMAC will try to add an arc to every blend (even straight lines).

I89 does not affect the behavior at inside corners, where the incoming and outgoing moves are always blended directly together, regardless of the change in directed angle.

Before V1.16 firmware, an arc was added to an outside corner if the change in directed angle were greater than 1°.

Example If it is desired that an arc only be added if the change in directed angle is greater than 45°, then I89 should be set to 0.707, because cos $\Delta\theta$ = cos 45^{o} = 0.707

See Also Cutter Radius Compensation (Writing a Motion Program)

I90 Minimum Arc Angle

Range Non-negative floating point

Units Semi-circles (π radians; 180 degrees)

Default 0 (sets 2^{-20})

Remarks I90 sets the threshold between a short arc and a full circle for CIRCLE mode moves in PMAC in all coordinate systems. I90 is expressed as an angle, with units that represent a fraction of a half-circle. It represents the smallest angle that can be covered by a programmed arc move.

Any programmed CIRCLE-mode move with an IJK-vector representation of the center which covers an angle smaller than I90 is executed as a full circle plus the programmed angle change. Any such move which covers an angle greater than I90 is executed as an arc smaller than a full circle.

The purpose of I90 is to support the circle programming standard that permits a full-circle move to be commanded simply by making the end point equal to the starting point (0 degree arc), yet allow for round-off errors.

Most users will be able to leave I90 at the default value of one-millionth of a semi-circle. This was formerly the fixed threshold value. However, some users may want to enlarge the threshold to compensate for round-off errors, particularly when using cutter-radius compensation in conjunction with full-circle moves. Remember that no arc covering an angle less than I90 can be executed.

If a full-circle move is commanded with cutter compensation on, and the blending from the previous move or into the next move creates a compensated outside corner without adding an arc (see I89), PMAC will extend the compensated move past a full circle. If I90 is too small, it may execute this as a very short arc, appearing to miss the move completely. I90 may have to be increased from its effective default value to cover this case.

For backward compatibility reasons, if I90 is set to 0, a threshold value of 2^{-20} (about one-millionth) of a semi-circle will be used.

See Also Cutter Radius Compensation
I-variable I89

I9x Motor x Second-Resolver Gear Ratio

Range 0 .. 4095

Units Primary-resolver turns per second-resolver turn

Default 0

Remarks I9x tells PMAC the gear ratio between the first (fine, or primary) and second (coarse or medium) resolvers for a double- or triple-resolver setup for Motor x. It is expressed as the

number of turns (electrical cycles) the first resolver makes in one full turn (electrical cycle) of the second resolver.

This parameter is used only during PMAC's power-up/reset cycle to establish absolute power-on servo position. Therefore, the parameter must be set, the value stored in EAROM with the **SAVE** command, and the card reset before it takes effect.

If there is no geared second resolver on Motor x, or if absolute power-on position is not desired, I9x should be set to zero. If Ix10 (for the primary resolver) is set to zero, I9x is not used. In a triple-resolver system, I9x must be set greater than zero in order for I8x (third-resolver gear ratio) to be used.

The second resolver must be connected to the next higher numbered R/D converter at the same multiplexer address than the first resolver. If there is a third resolver, it must be connected to the next higher numbered converter at the same multiplexer address than the second resolver. There can be up to eight R/D converters on two ACC-8D Option 7 boards at one multiplexer address.

If Ix10 is set up for an ACC-8D Option 9 Yaskawa encoder converter, I9x represents the counts per revolution (including x2 or x4 quadrature decode, if used) of the encoder; effectively it is the "gear ratio" between the encoder and the revolution counter.

Example Motor 1 has a double resolver with the fine resolver connected to the R/D converter at location 2 on an ACC-8D Option 7 board set to multiplexer address 4, and the coarse resolver, geared down at a 36:1 ratio from the fine resolver, connected to the R/D converter at location 3 on the same board. The following I-variable settings should be used:

```
I110=$020004 ....      ; Value of $0004 in low 16 bits specifies
.....                  ; multiplexer address 4; $02 in high 8 bits
.....                  ; specifies R/D at location 2 of this address
I91=36 .....           ; Specify 36 turns of fine resolver per turn of
.....                  ; coarse resolver; R/D must be at location 3
.....                  ; of multiplexer address 4
I81=0 .....            ; No third resolver
```

See Also Selecting the Position Loop Feedback (Setting Up a Motor)
I-Variables I8x, Ix03, Ix10, Ix81
ACC-8D Option 7 (R/D Converter) Manual

I99 Backlash Hysteresis

Range 0 .. 8,388,607

Units 1/16 count

Default 64 (= 4 counts)

Remarks This parameter controls the size of the direction reversal in motor commanded position that must occur on any motor before PMAC starts to add the programmed backlash (Ix86) in the direction of motion. The purpose of this variable is to allow the customer to ensure that a very small direction reversal (e.g. from the dithering of a master encoder) does not cause the backlash to "kick in". I99 thus provides a hysteresis in the backlash function.

The units of I99 are 1/16 of a count. Therefore this parameter must hold a value 16 times larger than the number of counts reversal at which backlash is introduced. For example, if backlash is to be introduced after 5 counts of reversal, I99 should be set to 80.

Before I99 was implemented, the backlash hysteresis was fixed at 4 counts, equivalent to the default I99 value of 64.

Example With a system in which one count of the master encoder creates 10 counts of movement in the slave motor, it is desired that a single count reversal of the master not trigger backlash reversal. Therefore the backlash hysteresis is set to 15 counts, and I99 is set to 15*16=240.

See Also Backlash Compensation (Setting Up a Motor)
I-Variables Ix85, Ix86

Motor x I-Variables

x = Motor Number (#x, x = 1 to 8)

Motor Definition I-Variables

Ix00 Motor x Activate

Range 0 .. 1

Units none

Default I100=1; I200 .. I800=0

Remarks Ix00 determines whether the Motor x is de-activated (=0) or activated (=1). If activated, position, servo, and trajectory calculations are done for the motor. An activated motor may be enabled – either in open or closed loop – or disabled (killed), depending on commands or events.

If Ix00 is 0, not even the position calculations for that motor are done, so a **P** command would not reflect position changes. Any PMAC motor not used should be de-activated, so PMAC does not waste time doing calculations for that motor. The fewer motors are activated, the faster the servo update time can be.

> ***Note:***
>
> Do not use Ix00 to kill a motor. Changing Ix00 from 1 to 0 leaves the motor outputs in whatever state they happened to be in at that moment.

See Also On-line commands **K**, **<CTRL-K>**, **A**, **<CTRL-A>**, **J/**

Ix01 Motor x PMAC-Commutation Enable

Range 0 .. 1

Units none

Default 0

Remarks Ix01 determines whether PMAC will perform commutation calculations for Motor x. If Ix01 is set to 0, PMAC will not perform commutation calculations for the motor, and it will compute only one output value for that motor (usually analog or pulse-and-direction). If a multi-phase motor is used, but is commutated in the amplifier, Ix01 should be set to 0.

If Ix01 is set to 1, PMAC will perform commutation calculations for Motor x. In this case, it will either compute two phase-current command outputs for the motor (if Ix82 = 0, disabling the current loop), usually analog outputs, or three phase-voltage command outputs (if Ix82 > 0, enabling the current loops), usually as PWM signals

See Also Setting Up PMAC Commutation
I-variables Ix70-Ix83.

Ix02 Motor x Command Output Address

Range Extended legal PMAC X and Y addresses

Units Extended legal PMAC X and Y addresses

Default

Motor	I-variable	PMAC(1)	PMAC2	PMAC2 Ultralite
Motor #1	I102	$C003	$C002	$C0A0
Motor #2	I202	$C002	$C00A	$C0A4
Motor #3	I302	$C00B	$C012	$C0A8
Motor #4	I402	$C00A	$C01A	$C0AC
Motor #5	I502	$C013	$C022	$C0B0
Motor #6	I602	$C012	$C02A	$C0B4
Motor #7	I702	$C01B	$C032	$C0B8
Motor #8	I802	$C01A	$C03A	$C0BC

Remarks Ix02 tells Motor x which register or registers to which it writes its command output values. It contains the address of this register or the first (lowest addresses) of these multiple registers. This determines which output lines transmit the command output signals. If bit 19 of Ix02 is set to 0 (default), this register is a Y-register; if bit 19 of Ix02 is set to 1, this register is an X-register. Almost all output registers on PMAC are Y-registers; the only common use of X-register outputs is in the Type 0 MACRO protocol. The exact function of Ix02 is dependent on the motor's mode of operation, as explained in the following sections.

No Commutation: If PMAC is not commutating Motor x (Ix01 = 0), only one command output value is calculated, which is written to the register at the address specified in Ix02.

For PMAC(1) systems, this output register is almost always a DAC analog output. The addresses of each DAC are shown in the following table.

Channel	Address	Channel	Address
DAC1	$C003	DAC9	$C023
DAC2	$C002	DAC10	$C022
DAC3	$C00B	DAC11	$C02B
DAC4	$C00A	DAC12	$C02A
DAC5	$C013	DAC13	$C033
DAC6	$C012	DAC14	$C032
DAC7	$C01B	DAC15	$C03B
DAC8	$C01A	DAC16	$C03A
Channels 9 – 16 are on an ACC-24P/V board.			

On PMAC(1) boards, if Ix01 is set to 0 and bit 16 of Ix02 is set to 1, then only the magnitude of the command is written to the register specified by Ix02 (e.g. I103=$1C003 to use DAC1 in this mode); the sign of the command is written to bit 14 of the flag register specified by Ix25, which is usually the AENA/DIR output. If this sign-and-magnitude mode is used, bit 16 of Ix25 should be set to 1 so this bit is not used for the amplifier-enable function. This mode is usually used with the ACC-8D Opt 2 voltage-to-frequency converter to generate pulse-and-direction signals for stepper-motor drives. Sign-and-magnitude mode is not available on PMAC2; for stepper applications it uses a fully digitally generated pulse train as described below.

In PMAC2 systems, if a single analog output is desired for the servo, it is usually the A DAC for the channel. The following table shows these addresses:

Channel	Address	Channel	Address
DAC1A	$C002	DAC9A	$C042
DAC2A	$C00A	DAC10A	$C04A
DAC3A	$C012	DAC11A	$C052
DAC4A	$C01A	DAC12A	$C05A
DAC5A	$C022	DAC13A	$C062
DAC6A	$C02A	DAC14A	$C06A
DAC7A	$C032	DAC15A	$C072
DAC8A	$C03A	DAC16A	$C07A
Channels 9 – 16 are on an ACC-24P/V2 board. For B-channel DAC registers, add 1 to the matching A-channel address			

When using a PMAC2 Ultralite board to command the servo over the MACRO ring, the command output is typically written to the MACRO node register 0. For the MACRO Type 1 protocol used with Delta Tau MACRO Stations, the addresses are shown in the following table:

Channel	Address	Channel	Address
Node 0 Reg. 0	$C0A0	Node 8 Reg. 0	$C0B0
Node 1 Reg. 0	$C0A4	Node 9 Reg. 0	$C0B4
Node 4 Reg. 0	$C0A8	Node 12 Reg. 0	$C0B8
Node 5 Reg. 0	$C0AC	Node 13 Reg. 0	$C0BC

One common application type for which the default value of Ix02 cannot be used is the direct pulse-and-direction output for stepper motor drives (PMAC2 only). This mode uses the C output register alone for each channel, and I9n6 for Channel n must be set to 2 or 3 to get pulse frequency output.

In this case, the following values should be used:

Channel	Address	Channel	Address
PWM1A	$C002	PWM9A	$C042
PWM2A	$C00A	PWM10A	$C04A
PWM3A	$C012	PWM11A	$C052
PWM4A	$C01A	PWM12A	$C05A
PWM5A	$C022	PWM13A	$C062
PWM6A	$C02A	PWM14A	$C06A
PWM7A	$C032	PWM15A	$C072
PWM8A	$C03A	PWM16A	$C07A
Channels 9 – 16 are on an ACC-24P/V2 board.			

Channel	Address	Channel	Address
PFM1	$C004	PFM9	$C044
PFM2	$C00C	PFM10	$C04C
PFM3	$C014	PFM11	$C054
PFM4	$C01C	PFM12	$C05C
PFM5	$C024	PFM13	$C064
PFM6	$C02C	PFM14	$C06C
PFM7	$C034	PFM15	$C074
PFM8	$C03C	PFM16	$C07C
Channels 9 – 16 are on an ACC-24P/V2 board			

When commanding pulse-and-direction from a PMAC Ultralite through a MACRO ring, use the address of Register 2 for the MACRO node, as shown in the following table:

Channel	Address	Channel	Address
Node 0 Reg. 0	$C0A2	Node 8 Reg. 0	$C0B2
Node 1 Reg. 0	$C0A6	Node 9 Reg. 0	$C0B6
Node 4 Reg. 0	$C0AA	Node 12 Reg. 0	$C0BA
Node 5 Reg. 0	$C0AE	Node 13 Reg. 0	$C0BE

Commutation, No Current Loop: If PMAC is commutating Motor x (Ix01 = 1), but not closing its current loop (Ix82 = 0), two command output values are calculated, which are written to the Y-register at the address specified in Ix02, plus the Y-register at the next higher address. Typically, these are two DAC output registers.

To use a pair of DACs on a PMAC(1), the address of the even-numbered DAC of the pair is used:

Channel	Address	Channel	Address
DAC1 & 2	$C002	DAC9 & 10	$C022
DAC3 & 4	$C00A	DAC11 & 12	$C02A
DAC5 & 6	$C012	DAC13 & 14	$C032
DAC7 & 8	$C01A	DAC15 & 16	$C03A
Channels 9 – 16 are on an ACC-24P/V board.			

To use a pair of DACs on a PMAC2, the address of the A-channel DAC is used to specify the use of both the A and B-channel DACs. The addresses used are the same as those for the case when the PMAC2 is not commutating the motor, whether directly or over MACRO.

In this mode, if bit 16 of Ix02 is set to 1 (e.g. I102=$1C002), then the PMAC will execute an open-loop commutation known as “direct microstepping” instead of the standard closed-loop commutation.

Commutation and Current Loop: If PMAC2 is commutating Motor x (Ix01 = 1) and closing its current loop (Ixx82 > 0), three command output values are calculated, which are written to the Y-register at the address specified in Ix02, plus the Y-registers at the next two higher addresses. This mode of operation is not supported on a PMAC(1).

In this mode, Ix02 typically specifies the A-channel output for the channel, which has been set up for PWM outputs (I9n6 = 0 for Channel n). The following table shows these addresses:

When commanding in this mode over the MACRO ring, the address specified is that of Register 0 for the MACRO node. The following table shows these addresses:

Channel	Address	Channel	Address
Node 0 Reg. 0	$C0A0	Node 8 Reg. 0	$C0B0
Node 1 Reg. 0	$C0A4	Node 9 Reg. 0	$C0B4
Node 4 Reg. 0	$C0A8	Node 12 Reg. 0	$C0B8
Node 5 Reg. 0	$C0AC	Node 13 Reg. 0	$C0BC

See Also Selecting the Output (Setting Up a Motor)
I-variables Ix01, Ix25, Ix70-Ix83
Memory-I/O registers Y:$C000-Y:$C03F

Ix03 Motor x Position Loop Feedback Address

Range Extended legal PMAC "X" addresses

Units Extended legal PMAC "X" addresses

Default

Variable	PMAC(1), PMAC2	Source with Default Table	PMAC2 Ultralite	Source with Default Table
I103	$0720	Converted ENC1	$0721	Converted Node 0
I203	$0721	Converted ENC2	$0723	Converted Node 1
I303	$0722	Converted ENC3	$0725	Converted Node 4
I403	$0723	Converted ENC4	$0727	Converted Node 5
I503	$0724	Converted ENC5	$0729	Converted Node 8
I603	$0725	Converted ENC6	$072B	Converted Node 9
I703	$0726	Converted ENC7	$072D	Converted Node 12
I803	$0727	Converted ENC8	$072F	Converted Node 13

Remarks Ix03 tells the PMAC where to look for its feedback to close the position loop for Motor x. Usually it points to an entry in the Encoder Conversion Table, where the values from the encoder counter registers have been processed at the beginning of each servo cycle (possibly to include sub-count data). This table starts at address $0720 and continues until address $073F. It is shipped from the factory configured as shown in the default table above.

For a motor with dual feedback (motor and load), use Ix03 to point to the encoder on the load, and Ix04 to point to the encoder on the motor.

If the position loop feedback device is the same device as is used for commutation (with PMAC doing the commutation), then it must also be specified for commutation with Ix83. However, Ix83 should specify the address of the encoder counter itself, not the converted data of the table.

Hardware Home Position Capture: The source address of the position information occupies bits 0 to 15 of Ix03 (range $0000 to $FFFF, or 0 to 65535). With bit 16 equal to zero – the normal case – position capture on homing is done with the hardware capture register associated with the flag inputs pointed to by Ix25. In this case, it is important to match the encoder number, the address pointed to with Ix03, with the flag number, the address pointed to with Ix03 (e.g. ENC1 – CHA1 & CHB1 – with HMFL1 and LIM1).

Software Home Position Capture: If bit 16 (value 65536) is set to one, the position capture on homing is done through software, and the position source does not have to match the input flag source. This is particularly important for parallel-data position feedback, such as from a laser interferometer (which is incremental data and requires homing). For example, if motor #1 used parallel feedback from a laser interferometer processed as the first (triple) entry in the conversion table, the key I-variables would be:

I103=$10722 I125=$C000

This would permit homing on interferometer data with HMFL1 triggering.

Note:

In the extended version, it is obviously easier to specify this parameter in hexadecimal form. With I9 at 2 or 3, the value of this variable will be reported back to the host in hexadecimal form.

Capture on following error: If bit 17 of Ix03 is set to 1, then the trigger for position capture of this motor is a true state on the warning following error status bit for the motor. If bit 17 is at the default of 0, the trigger for position capture is the capture flag of the flag registers as set by Ix25. The trigger is used in two types of moves: homing search moves and programmed move-until-triggers. If bit 17 is set to 1, the triggered position must be software captured, so bit 16 must also be set to 1 to specify software captured bit position.

Hardware Capture with Normal-Resolution Feedback: If bit 18 of Ix03 is set to its default value of 0 when hardware position capture is used in a triggered move such as a homing-search move, the captured data (whether whole-count only or including sub-count data) is processed to match servo feedback of "normal" resolution (5 bits of fractional count data per hardware whole count). This setting is appropriate for digital quadrature feedback or for "low-resolution" interpolation of a sinusoidal encoder.

Hardware Capture with High-Resolution Interpolated Feedback: If bit 18 (value $40000, or 262,144) is set to 1 when hardware position capture is used in a triggered move, the captured data (whether whole-count only or including sub-count data) is processed to match servo feedback of "high" resolution (10 bits of fractional count data per hardware whole count). This setting is appropriate for "high-resolution" interpolation of a sinusoidal encoder through an ACC-51x interpolator.

Whole-Count Capture: If bit 19 of Ix03 is set to 0 when hardware position capture is used in a triggered move such as a homing-search move, only the whole-count captured position register is used to establish the trigger position. This setting must be used on PMAC(1) controllers, and on PMAC2 controllers with Servo ICs older than Revision "D" (Revision "D" ICs started shipping in early 2002).

Sub-Count Capture: If bit 19 (value $80000, or 524,288) is set to 1 when hardware position capture is used in a triggered move, both the whole-count captured position register and the estimated sub-count position register are used to establish the trigger position. This setting can only be used on PMAC2 controllers with Servo ICs of Revision "D" or newer. I9n9 for the Channel "n" used for the capture must be set to 1 to enable the hardware sub-count estimation. This setting is typically used for registration or probing triggered moves with interpolated sinusoidal encoder feedback. (Even with interpolated sinusoidal encoder feedback, homing-search moves will probably be done without sub-count captured data, to force a home position referenced to one of the four "zero-crossing" positions of the sine/cosine signals.)

See Also Selecting the Position Loop Feedback (Setting Up a Motor)
Encoder Conversion Table (Setting Up a Motor)
I-variables Ix04, Ix05, Ix25, Ix83.

Ix04 Motor x Velocity Loop Feedback Address

Range Legal PMAC X addresses

Units Legal PMAC X addresses

Default Same as Ix03

Remarks Ix04 holds the address of the position feedback device that PMAC uses for its velocity-loop feedback information. For a motor with only a single feedback device (the usual case), this must be the same as Ix03. For a motor with dual feedback (motor and load), use Ix04 to point to the encoder on the motor, and Ix03 to point to the encoder on the load.

If the velocity-loop feedback device is the same device as is used for commutation (if PMAC is doing the commutation), then both Ix04 and Ix83 (commutation feedback address) must reference the same device. However, Ix04 typically points to the converted data – a register in the Encoder Conversion table – while Ix83 must point directly to the DSPGATE encoder register.

The instructions for setting this parameter are identical to those for Ix03, except that there are no address extension bits.

Note:

When planning which channels to use when connecting the position and velocity encoders, remember that the channel pointed to by Ix25 is used for the Overtravel, Amplifier Fault, and Home Flag inputs.

See Also Selecting the Velocity-Loop Feedback (Setting Up a Motor)
Encoder Conversion Table (Setting Up a Motor)
I-variables Ix03, Ix05, Ix25, Ix83.

Ix05 Motor x Master (Handwheel) Position Address

Range Legal PMAC X addresses

Units Legal PMAC X addresses

Default $073F (1855) (= zero register at end of conversion table)

Remarks Ix05 tells the PMAC where to look for the position of the master, or handwheel, encoder for Motor x. Usually this is an entry in the Encoder Conversion Table that holds processed information from an encoder channel. The instructions for setting this parameter are identical to those for Ix03, except the extended bits mean different things. The default value permits handwheel input from the JPAN connector (jumpered into the ENC2 counter with E22 and E23).

Following Modes: The source address of the position information occupies bits 0 to 15 of Ix05 (range $0000 to $FFFF, or 0 to 65535). With bit 16 equal to zero – the normal case – position following is done with the actual position reported for the motor reflecting the change due to the following. With bit 16 – value 65536 – equal to one, the actual position reported for the motor does not reflect the change due to the following ("offset" mode). This mode can be useful for part offsets, and for superimposing programmed and following moves. For example, to have motor #1 following encoder 2 in offset mode, I105 should be set to $10721.

Note:

In the extended version, it is easier to specify this parameter in hexadecimal form. With I9 at 2 or 3, the value of this variable will be reported back to the host in hexadecimal form.

Note:

It is important not to have the same source be both the master and the feedback for an individual motor. If this is the case, with Ix06=1 to enable following, the motor will run away (it is like a puppy chasing its tail – it cannot catch up to its desired position, because its desired position keeps moving ahead of it).

If you want to ensure that following cannot occur by accident, you may want to change Ix05 so it points to a register that cannot change. This way, even if the following function gets turned on, for instance by the motor selector inputs on the JPAN connector, no following can occur. The best registers to use for this purpose are the unused ones at the end of the conversion table. With the default table setup, you can choose any register between $072A and $073F (1834 to 1855 decimal). If you extend the table, choose a register between the end of the table and $073F.

See Also Selecting the Master Position Source (Setting Up a Motor)
Encoder Conversion Table (Setting Up a Motor)
Position Following (Synchronizing PMAC to External Events)
I-variables Ix03, Ix04, Ix06, Ix08, Ix09.

Ix06 Motor x Master (Handwheel) Following Enable

Range 0 .. 1

Units none

Default 0

Remarks Ix06 disables or enables Motor x's position following function. A value of 0 means disabled; a value of 1 means enabled. Following mode is specified by high bits of Ix05.

> ***Note:***
>
> This parameter can be changed on-line in a PMAC(1) through hardware inputs on the JPAN connector. The FPDn/ motor/coordinate-system select lines (low-true BCD-coded) can turn Ix06 on and off. On power-up or reset, if I2 was saved as zero, Ix06 for the selected motor is set to one and Ix06 for all other motors is set to zero regardless of the values that were saved. When the select switch is changed, Ix06 for the de-selected motor is set to zero and Ix06 for the selected motor is set to 1.

See Also Position Following (Synchronizing PMAC to External Events)
I-variables I2, Ix05, Ix07, Ix08
Control Panel Inputs (Connecting PMAC to the Machine)

Ix07 Motor x Master (Handwheel) Scale Factor

Range -8,388,608 .. 8,388,607

Units none

Default 96

Remarks Ix08 controls with what scaling the master (handwheel) encoder gets extended into the full-length register. In combination with Ix08, it also controls the following ratio of Motor x (delta-motor-x = [Ix07/Ix08] * delta-handwheel-x) for position following (electronic gearing). For following, Ix07 and Ix08 can be thought of as the number of teeth on meshing gears in a mechanical coupling.

Ix07 can be changed on the fly to permit real-time changing of the following ratio, but Ix08 may not.

See Also I-variables Ix05, Ix06, Ix08
Position Following (Synchronizing PMAC to External Events)

Ix08 Motor x Position Scale Factor

Range 0 .. 8,388,607

Units none

Default 96

Remarks Ix08 controls how the position encoder counter gets extended into the full-length register. For most purposes, this is transparent to the user and does not need to be changed from the default.

There are two reasons that the user might want to change this from the default value. First, because it is involved in the "gear ratio" of the position following function – the ratio is Ix07/Ix08 – this might be changed (usually raised) to get a more precise ratio.

The second reason to change this parameter (usually lowering it) is to prevent internal saturation at very high gains or count rates (velocity). PMAC's filter will saturate when the velocity in counts/sec multiplied by Ix08 exceeds 768M (805,306,368). This happens only in very rare applications – the count rate must exceed 8.3 million counts per second before the default value of Ix08 gives a problem.

> ***Note:***
>
> When changing this parameter, make sure the motor is killed (disabled). Otherwise, a sudden jump will occur, because the internal position registers will have changed. This means that this parameter should not be changed in the middle of an application. If a real-time change in the position-following "gear ratio" is desired, Ix07 should be changed.

In most practical cases, Ix08 should not be set above 1000 because higher values can make the servo filter saturate too easily. If Ix08 is changed, Ix30 should be changed inversely to keep the same servo performance (e.g. if Ix08 is doubled, Ix30 should be halved).

See Also Position Following (Synchronizing PMAC to External Events)
I-variables Ix05, Ix06, Ix07, Ix09, Ix30

Ix09 Motor x Velocity Loop Scale Factor

Range 0 .. 8,388,607

Units none

Default 96

Remarks Ix09 controls how the encoder counter used to close the velocity servo loop gets extended into the full-length register. For most purposes, this is transparent to the user and does not need to be changed from the default. This parameter should not be changed in the middle of an application, because it scales many internal values. If the same sensor is used to close both the position and velocity loops (Ix03), Ix09 should be set equal to Ix08.

If different sensors are used, Ix09 should be set such that the ratio of Ix09 to Ix08 is inversely proportional to the ratio of the velocity sensor resolution (at the load) to the position sensor resolution.

Example If a 5000 line/inch (20,000 cts/in) linear encoder is used for position feedback, and a 500 line/rev (2000 cts/rev) rotary encoder is used for velocity loop feedback, and there is a 5-pitch screw, the effective resolution of the velocity encoder is 10,000 cts/in (2000x5), half of the position sensor resolution, so Ix09 should be set to twice Ix08.

If the value computed this way for Ix09 does not come to an integer, use the nearest integer value.

See Also I-variables Ix03, Ix04, Ix08, Ix31
Dual-Feedback Systems (Setting Up a Motor)

Ix10 Motor x Power-Up Servo Position Address

Range $000000 - $FFFFFF

Units Extended PMAC Addresses

Default 0

Remarks Ix10 controls whether PMAC reads an absolute position sensor for Motor x on power-up/reset and/or with the **$*** command. If an absolute position read is to be done, Ix10 specifies what register is read for that absolute position data and how the data in this register is interpreted.

If Ix10 is set to 0, no absolute power-on/reset position read is performed. The power-on/reset position is considered to be zero, even if an absolute sensor reporting a non-zero value is used. Ix10 should be set to 0 when an incremental position sensor is used; typically a homing search move is then executed to establish a position reference.

If Ix10 is set to a non-zero value, an absolute position read is performed for Motor x at power-on/reset (unless Bit 2 of Ix80 is set to 1), or on the **$*** command, from the register whose address is specified in Ix10. The motor's position is set to the value read from the sensor minus the Ix26 "home" offset value.

Ix10 consists of two parts. The low 16 bits, represented by four hex digits, contain the address of the register containing the power-on position information, either a PMAC memory-I/O address, an address on the multiplexer ("thumbwheel") port, or the *number* of the MACRO node on the PMAC, depending on the setting of the high 8 bits. The high 8 bits, represented by two hex digits, specify how to read the information at this address.

Note:

It is easier to specify this parameter in hexadecimal form ($ prefix). If I9 is set to 2 or 3, the value of this variable will be reported back to the host in hexadecimal form.

The possible values of Bits 16 – 23 of Ix10 and the absolute position feedback devices they reference are summarized in the following table:

Ix10 Value Range	Absolute Position Source	Ix10 Address Type	Format
$00xxxx - $07xxxx	ACC-8D Opt 7 R/D Converter	Multiplexer Port	Unsigned
$08xxxx - $30xxxx	Parallel Data Y-Register	PMAC Memory-I/O	Unsigned
$31xxxx	ACC-28 A/D Converter	PMAC Memory-I/O	Unsigned
$32xxxx	ACC-49 Sanyo Abs. Encoder	PMAC Memory-I/O	Unsigned
$48xxxx - $70xxxx	Parallel Data X-Register	PMAC Memory-I/O	Unsigned
$71xxxx	ACC-8D Opt 9 Yaskawa Abs. Enc.	Multiplexer Port	Unsigned
$72xxxx	MACRO Station Yaskawa Abs. Enc.	MACRO Node Number	Unsigned
$73xxxx	MACRO Station R/D Converter	MACRO Node Number	Unsigned
$74xxxx	MACRO Station Parallel Read	MACRO Node Number	Unsigned
$75xxxx	EnDat Data Read (Geo PMAC)	Geo PMAC	Unsigned
$80xxxx - $87xxxx	ACC-8D Opt 7 R/D Converter	Multiplexer Port	Signed
$88xxxx - $B0xxxx	Parallel Data Y-Register	PMAC Memory-I/O	Signed
$B1xxxx	ACC-28 A/D Converter	PMAC Memory-I/O	Signed
$B2xxxx	ACC-49 Sanyo Abs. Encoder	PMAC Memory-I/O	Signed
$C8xxxx - $F0xxxx	Parallel Data X-Register	PMAC Memory-I/O	Signed
$F1xxxx	ACC-8D Opt 9 Yaskawa Abs. Enc.	Multiplexer Port	Signed
$F2xxxx	MACRO Station Yaskawa Abs. Enc.	MACRO Node Number	Signed

$F3xxxx	MACRO Station R/D Converter	MACRO Node Number	Signed
$F4xxxx	MACRO Station Parallel Read	MACRO Node Number	Signed
$F4xxxx	EnDat Data Read (Geo PMAC)	Geo PMAC	Signed

The following section provides details for each type of position feedback.

R/D Converter: If Ix10 contains a value from $0000xx to $0700xx, or from $8000xx to $8700xx, Motor x will expect its absolute power-on position from an ACC-8D Opt. 7 R/D converter board. The low 8 bits (last 2 hex digits) of Ix10 should contain the address of the board on the multiplexer port, as set by the DIP switches on the board.

The first hex digit of Ix10, which can take a value of 0 or 8 in this mode, specifies whether the position is interpreted as an unsigned value (1st digit = 0) or as a signed value (1st digit = 8).

The second hex digit of Ix10, which can take a value from 0 to 7 in this mode, specifies the number of the individual R/D converter at that multiplexer port address. The following table shows the Ix10 values for this mode and the R/D converter each specifies at the 'xx' multiplexer-port address:

Ix10 Value for Unsigned Position	Ix10 Value for Signed Position	ACC-8D Opt. 7 SW1-1 Setting	# of R/D Converter on ACC-8D Opt. 7
$0000xx*	$8000xx	CLOSED (0)	1
$0100xx	$8100xx	CLOSED (0)	2
$0200xx	$8200xx	CLOSED (0)	3
$0300xx	$8300xx	CLOSED (0)	4
$0400xx	$8400xx	OPEN (1)	1
$0500xx	$8500xx	OPEN (1)	2
$0600xx	$8600xx	OPEN (1)	3
$0700xx	$8700xx	OPEN (1)	4
*If 'xx' is '00', the fourth hex digit should be set to 1(making Ix10=$000100); otherwise no absolute position will be read because Ix10=0.			

If I9x is set greater than 0, the next higher numbered R/D converter at the same multiplexer port address is also read and treated as a geared-down resolver, with I9x specifying the gear ratio. I8x is also set greater than 0, the following R/D converter at the same multiplexer port address is read and treated as a third resolver geared down from the second, with I8x specifying that gear ratio.

The following table shows the values of Ix10 for the multiplexer port addresses for the ACC-8D Opt. 7 that can be used:

Board Mux. Addr.	Ix10	Board Mux. Addr	Ix10	Board Mux. Addr	Ix10	Board Mux. Addr	Ix10
0	$0n0000	64	$0n0040	128	$0n0080	192	$0n00C0
8	$0n0008	72	$0n0048	136	$0n0088	200	$0n00C8
16	$0n0010	80	$0n0050	144	$0n0090	208	$0n00D0
24	$0n0018	88	$0n0058	152	$0n0098	216	$0n00D8
32	$0n0020	96	$0n0060	160	$0n00A0	224	$0n00E0
40	$0n0028	104	$0n0068	168	$0n00A8	232	$0n00E8
48	$0n0030	112	$0n0070	176	$0n00B0	240	$0n00F0
56	$0n0038	120	$0n0078	184	$0n00B8	248	$0n00F8
'n' is a digit from 0 to 7 specifying the converter number at that address * If 'n' is 0 and the multiplexer address is 0, the 4th hex digit should be set to 1, making Ix10=$000100; otherwise with Ix10=0, no absolute position would be read.							

Parallel Data Read: If Ix10 contains a value from $08xxxx to $30xxxx, from $48xxxx to $70xxxx, from $88xxxx to $B0xxxx, or from $C8xxxx to $F0xxxx, Motor x will do a parallel data read of the PMAC memory or I/O register at address 'xxxx'.

In this mode, bits 16 to 21 of Ix10 specify the number of bits to be read, starting with bit 0 at the specified address. In this mode, they can take a value from $08 to $30 (8 to 48). If the number of bits is greater than 24, the high bits are read from the register at the next higher-numbered address.

In this mode, bit 22 of Ix10 specifies whether a Y-register is to be read, or an X-register. A value of 0 in this bit specifies a Y-register; a value of 1 specifies an X-register. Almost all common sources of absolute position information are located in Y-registers, so this digit is almost always 0.

Bit 23 of Ix10 specifies whether the position is interpreted as an unsigned or a signed value. If the bit is set to 0, it is interpreted as an unsigned value, if the bit is 1, it is interpreted as a signed value.

Combining these components, Ix10 values in this mode can be summarized as:

- $08xxxx - $30xxxx: Parallel Y-register read, unsigned value, 8 to 48 bits
- $48xxxx - $70xxxx: Parallel X-register read, unsigned value, 8 to 48 bits
- $88xxxx - $B0xxxx: Parallel Y-register read, signed value, 8 to 48 bits
- $C8xxxx - $F0xxxx: Parallel X-register read, signed value, 8 to 48 bits

The following table shows Ix10 values for parallel data read through an ACC-14 board.

Register	Ix10	Register	Ix10
1st ACC-14D/V Port A	$xxFFD0	4th ACC-14D/V Port A	$xxFFE8
1st ACC-14D/V Port B	$xxFFD1	4th ACC-14D/V Port B	$xxFFE9
2nd ACC-14D/V Port A	$xxFFD8	5th ACC-14D/V Port A	$xxFFF0
2nd ACC-14D/V Port B	$xxFFD9	5th ACC-14D/V Port B	$xxFFF1
3rd ACC-14D/V Port A	$xxFFE0	6th ACC-14D/V Port A	$xxFFF8
3rd ACC-14D/V Port B	$xxFFE1	6th ACC-14D/V Port B	$xxFFF9
'xx' represent the first two digits, which control bit width, and signed vs. unsigned data. ACC-14 boards are always Y-addresses			

For reading MLDT absolute position from a PMAC timer register, the first two hex digits of Ix10 are set to $58. Bits 16 – 21 are set to $18 to specify a 24-bit register; bit 22 is set to 1 ($40) to specify an X-register, and bit 23 is set to 0 to specify an unsigned value.

The following table shows Ix10 values for reading ACC-29 MLDT timer registers on a PMAC(1) as parallel data:

Channel	Ix10	Channel	Ix10
9	$58C020	13	$58C030
10	$58C024	14	$58C034
11	$58C028	15	$58C038
12	$58C02C	16	$58C03C

The following table shows Ix10 values for reading PMAC2 built-in MLDT timer registers:

Channel	Ix10	Channel	Ix10
1	$58C000	5	$58C020
2	$58C008	6	$58C028
3	$58C010	7	$58C030
4	$58C018	8	$58C038

ACC-28 A/D Converter Read: If Ix10 is set to $31xxxx or $B1xxxx, Motor x will expect its power-on position in the upper 16 bits of the PMAC Y-memory or I/O register specified by 'xxxx'. This format is intended for ACC-28 A/D converters.

Bit 23 of Ix10 specifies whether the position is interpreted as an unsigned or a signed value. If the bit is set to 0, it is interpreted as an unsigned value, if the bit is 1, it is interpreted as a signed value. Because ACC-28A produces signed values, Ix10 should be set to $B1xxxx when using ACC-28A. ACC-28B produces unsigned values, so Ix10 should be set to $31xxxx when using ACC-28B.

The following tables show Ix10 values for ACC-28A/B on PMAC(1) and Ix10 values for ACC-28B through PMAC2, respectively.

Channel	Ix10 for ACC-28A	Ix10 for ACC-28B	Channel	Ix10 for ACC-28A	Ix10 for ACC-28B
1	$B1C006	$31C006	9	$B1C026	$31C026
2	$B1C007	$31C007	10	$B1C027	$31C027
3	$B1C00E	$31C00E	11	$B1C02E	$31C02E
4	$B1C00F	$31C00F	12	$B1C02F	$31C02F
5	$B1C016	$31C016	13	$B1C036	$31C036
6	$B1C017	$31C017	14	$B1C037	$31C037
7	$B1C01E	$31C01E	15	$B1C03E	$31C03E
8	$B1C01F	$31C01F	16	$B1C03F	$31C03F
Channels 9 through 16 are brought in through an ACC-24 board					

Channel	Ix10 for ADC A	Ix10 for ADC B	Channel	Ix10 for ADC A	Ix10 for ADC B
1	$31C005	$31C006	9	$31C045	$31C046
2	$31C00D	$31C00E	10	$31C04D	$31C04E
3	$31C015	$31C016	11	$31C055	$31C056
4	$31C01D	$31C01E	12	$31C05D	$31C05E
5	$31C025	$31C026	13	$31C065	$31C066
6	$31C02D	$31C02E	14	$31C06D	$31C06E
7	$31C035	$31C036	15	$31C075	$31C076
8	$31C03D	$31C03E	16	$31C07D	$31C07E
Channels 9 through 16 are brought in through an ACC-24P/V2 board.					

Sanyo Absolute Encoder Read: If Ix10 is set to $32xxxx or $B2xxxx, Motor x will expect its power-on position from the ACC-49 Sanyo Absolute Encoder converter board at the PMAC Y-address specified by 'xxxx'.

Bit 23 of Ix10 specifies whether the position is interpreted as an unsigned value (Bit 23 = 0, making the first hex digit a 3) or as a signed value (Bit 23 = 1, making the first hex digit a B). Set Ix10 to $32xxxx for unsigned, or to $B2xxxx for signed.

The following table lists the possible values of Ix10 for the ACC-49:

Enc. # on Board	Ix10 for E1 ON	Ix10 for E2 ON	Ix10 for E3 ON
Enc. 1	$m2FFD0	$m2FFD8	$m2FFE0
Enc. 2	$m2FFD4	$m2FFDC	$m2FFE4
Enc. # on Board	**Ix10 for E4 ON**	**Ix10 for E5 ON**	**Ix10 for E6 ON**
Enc. 3	$m2FFE8	$m2FFF0	$m2FFF8
Enc. 4	$m2FFEC	$m2FFF4	$m2FFFC
m is 3 or B depending on whether the data is to be interpreted as an unsigned or signed quantity.			

Yaskawa Absolute Encoder Read: If Ix10 is set to $7100xx or $F100xx, Motor x will expect

its power-on position from the Yaskawa Absolute Encoder converter board at the multiplexer port address specified by 'xx'.

Bit 23 of Ix10 specifies whether the position is interpreted as an unsigned value (Bit 23 = 0, making the first hex digit a 7) or as a signed value (Bit 23 = 1, making the first hex digit an F). Set Ix10 to $7100xx for unsigned, or to $F100xx for signed.

In this mode, I9x specifies the number of bits per revolution for a single turn of the Yaskawa absolute encoder. (For example, with 8192 counts per revolution, there are 13 bits per revolution.) It must be set greater than 0 to use the multi-turn absolute capability of this encoder.

The following table shows the values of Ix10 for the ACC-8D Option 9:

Board Mux. Addr.	Ix10 for Enc. 1	Ix10 for Enc. 2	Ix10 for Enc. 3	Ix10 for Enc. 4	Board Mux. Addr.	Ix10 for Enc. 1	Ix10 for Enc. 2	Ix10 for Enc. 3	Ix10 for Enc. 4
0	$m10000	$m10002	$m10004	$m10006	128	$m10080	$m10082	$m10084	$m10086
8	$m10008	$m1000A	$m1000C	$m1000E	136	$m10088	$m1008A	$m1008C	$m1008E
16	$m10010	$m10012	$m10014	$m10016	144	$m10090	$m10092	$m10094	$m10096
24	$m10018	$m1001A	$m1001C	$m1001E	152	$m10098	$m1009A	$m1009C	$m1009E
32	$m10020	$m10022	$m10024	$m10026	160	$m100A0	$m100A2	$m100A4	$m100A6
40	$m10028	$m1002A	$m1002C	$m1002E	168	$m100A8	$m100AA	$m100AC	$m100AE
48	$m10030	$m10032	$m10034	$m10036	176	$m100B0	$m100B2	$m100B4	$m100B6
56	$m10038	$m1003A	$m1003C	$m1003E	184	$m100B8	$m100BA	$m100BC	$m100BE
64	$m10040	$m10042	$m10044	$m10046	192	$m100C0	$m100C2	$m100C4	$m100C6
72	$m10048	$m1004A	$m1004C	$m1004E	200	$m100C8	$m100CA	$m100CC	$m100CE
80	$m10050	$m10052	$m10054	$m10056	208	$m100D0	$m100D2	$m100D4	$m100D6
88	$m10058	$m1005A	$m1005C	$m1005E	216	$m100D8	$m100DA	$m100DC	$m100DE
96	$m10060	$m10062	$m10064	$m10066	224	$m100E0	$m100E2	$m100E4	$m100E6
104	$m10068	$m1006A	$m1006C	$m1006E	232	$m100E8	$m100EA	$m100EC	$m100EE
112	$m10070	$m10072	$m10074	$m10076	240	$m100F0	$m100F2	$m100F4	$m100F6
120	$m10078	$m1007A	$m1007C	$m1007E	248	$m100F8	$m100FA	$m100FC	$m100FE
m is 7 or F, depending on whether the data is to be interpreted as an unsigned or signed quantity.									

MACRO Station Yaskawa Absolute Encoder Read: If Ix10 is set to $72000n or $F2000n, Motor x will expect its power-on position from a Yaskawa Absolute Encoder through a MACRO Station. In this mode, 'n' specifies the MACRO node number at which the position value will be read by PMAC itself. Set-up variable MI11x for the MACRO Station tells the Station how to read the Yaskawa Encoder converter connected to its own multiplexer port or serial port.

Bit 23 of Ix10 specifies whether the position is interpreted as an unsigned value (Bit 23 = 0, making the first hex digit a 7) or as a signed value (Bit 23 = 1, making the first hex digit an F). Set Ix10 to $72000n for unsigned, or to $F2000n for signed.

In this mode, I9x specifies the number of bits per revolution for a single turn of the Yaskawa absolute encoder. (For example, with 8192 counts per revolution, there are 13 bits per revolution.) It must be set greater than 0 to use the multi-turn absolute capability of this encoder.

MACRO Station R/D Converter Read: If Ix10 is set to $73000n or $F3000n, Motor x will expect its power-on position from an ACC-8D Opt 7 R/D converter through a MACRO Station or compatible device. In this mode, 'n' specifies the MACRO node number at which PMAC will read the position value itself. Set-up variable MI11x for the matching node on the MACRO Station tells the Station how to read the R/D converter connected to its own multiplexer port.

Bit 23 of Ix10 specifies whether the position is interpreted as an unsigned value (Bit 23 = 0, making the first hex digit a 7) or as a signed value (Bit 23 = 1, making the first hex digit an F). Set Ix10 to $73000n for unsigned, or to $F3000n for signed.

If I9x is set greater than 0, the next higher numbered R/D converter at the same multiplexer

port address is also read and treated as a geared-down resolver, with I9x specifying the gear ratio. If I8x is also set greater than 0, the following R/D converter at the same multiplexer port address is read and treated as a third resolver geared down from the second, with I8x specifying that gear ratio.

MACRO Station Parallel Data Read: If Ix10 is set to $74000n or $F4000n, Motor x will expect its power-on position from a parallel data source through a MACRO Station or compatible device. In this mode, 'n' specifies the MACRO node number at which PMAC will read the position value itself. Set-up variable MI11x for the matching node on the MACRO Station tells the Station how to read the parallel data source connected to it.

Bit 23 of Ix10 specifies whether the position is interpreted as an unsigned value (Bit 23 = 0, making the first hex digit a 7) or as a signed value (Bit 23 = 1, making the first hex digit an F). Set Ix10 to $74000n for unsigned, or to $F4000n for signed.

EnDat Data Read (Geo PMAC only): If Ix10 is set to $75wxyz or $F5wxyz on a Geo PMAC, Motor x will expect its power-on servo position from the optional EnDat interface that can be built in to the Geo PMAC.

Bit 23 of Ix10 specifies whether the position is interpreted as an unsigned value (Bit 23 = 0, making the first hex digit a 7) or as a signed value (Bit 23 = 1, making the first hex digit an F). Set Ix10 to $75wxyz for unsigned, or to $F5wxyz for signed.

The third and fourth hex digits (wx) specify the number of bits of EnDat absolute position to be read, as a hex value. For example, if 20 bits were to be read, these two hex digits would be set to 14.

The fifth hex digit (y) specifies the shift of the data read, permitting the user to match the resolution properly with that of the ongoing position. The data is first shifted left 10 bits, then shifted right by "y" bits, so "y" should be set to (NetRightShift + 10). The object is to end up with data whose LSB is equal to one quadrature count (1/4-line) of the encoder. Most commonly, the data comes in with the LSB equal to 1/4 of a quadrature count (1/16-line), requiring a net right shift of 2 bits, so "y" should be a hex digit of "C" (12 decimal).

The sixth hex digit (z) specifies the channel number used, whether the data is negated, and whether it should be matched to ongoing digital quadrature or analog sinusoidal feedback. Bits 1 and 0 together express the channel number minus one as a value from 0 to 3 (so Channel 1 to 4). Bit 2 is set to 0 if the ongoing feedback is analog sinusoidal processed through the high-resolution conversion (format $F) in the conversion table, or to 1 if the ongoing feedback is digital quadrature. Bit 3 is set to 0 to use the data without negation, or to 1 to negate the data before using. Negating the data reverses the direction sense; this control is used to match the direction sense of the ongoing feedback as set by I9n0.

Presently, Channels 1 and 2 are supported, and EnDat is almost always used with interpolated sinusoidal ongoing feedback, so "z" is set to 0 or 8 for Channel 1 (regular or negated, respectively), or to 1 or 9 for Channel 2 (regular or negated, respectively).

Motor Safety I-Variables

Ix11 Motor x Fatal (Shutdown) Following Error Limit

Range 0 .. 8,388,607

Units 1/16 Count

Default 32000 (2000 counts)

Remarks Ix11 sets the magnitude of the following error for Motor x at which operation will shut down. When the magnitude of the following error exceeds Ix11, Motor x is disabled

(killed). If the motor's coordinate system is executing a program at the time, the program is aborted. It is optional whether other PMAC motors are disabled when this motor exceeds its following error limit; bits 21 and 22 of Ix25 control what happens to the other motor (the default is that all PMAC motors are disabled).

A status bit for the motor, and one for the coordinate system (if the motor is in one) are set. If this coordinate system is hardware-selected on JPAN (with I2=0), or software-addressed by the host (with I2=1), the ERLD/ output on JPAN, and the EROR input to the interrupt controller (except for PMAC-VME) are triggered.

Setting Ix11 to zero disables the fatal following error limit for the motor. This may be desirable during initial development work, but it is *strongly discouraged* in an actual application. A fatal following error limit is a very important protection against various types of faults, such as loss of feedback, that cannot be detected directly, and that can cause severe damage to people and equipment.

Note:

The units of Ix11 are 1/16 of a count. Therefore this parameter must hold a value 16 times larger than the number of counts at which the limit will occur. For example, if the limit is to be 1000 counts, Ix11 should be set to 16,000.

See Also

I-variables I2, Ix12, Ix25
Following Error Limits, Amplifier Fault (Making Your Application Safe)
Control Panel Outputs (Connecting PMAC to the Machine)
Using Interrupts (Writing a Host Communications Program)
Memory registers Y:$0814, Y:08D4, etc., Y:$0817, Y:$08D7, etc.
On-line commands **?**, **??**.

Ix12 Motor x Warning Following Error Limit

Range 0 .. 8,388,607

Units 1/16 Counts

Default 16000 (1000 counts)

Remarks Ix12 sets the magnitude of the following error for Motor x at which a warning flag goes true. If this limit is exceeded, status bits are set for the motor and the motor's coordinate system (if any). The coordinate system status bit is the logical OR of the status bits of all the motors in the coordinate system.

Setting this parameter to zero disables the warning following error limit function. If this parameter is set greater than the fatal following error limit, the warning status bit will never go true, because the fatal limit will disable the motor first.

If bit 17 of Ix03 is set to 1, the motor can be triggered for homing search moves, jog-until-trigger moves, and motion program move-until-trigger moves when the following error exceeds Ix12. This is known as torque-mode triggering, because the trigger will occur at a torque level corresponding to the Ix12 limit.

At any given time, one coordinate system's status bit can be output to several places; which system depends on what coordinate system is hardware-selected on the panel input port if I2=0, or what coordinate system is software-addressed from the host (&n) if I2=1. The outputs that work in this way are F1LD/ (pin 23 on connector J2), F1ER (line IR3 into the programmable interrupt controller (PIC) on PMAC-PC, line IR6 into the PIC on PMAC-STD) and, if E28 connects pins 1 and 2, FEFCO/ (on the JMACH connectors).

Note:

The units of Ix12 are 1/16 of a count. Therefore this parameter must hold a value 16 times larger than the number of counts at which the limit will occur. For example, if the limit is to be 1000 counts, Ix12 should be set to 16,000.

See Also Control-panel (JPAN) output F1LD/, Interrupt line F1ER/
Following Error Limits (Making Your Application Safe)
Control Panel Outputs (Connecting PMAC to the Machine)
Torque-Mode Triggering (Basic Motor Moves)
Using Interrupts (Writing a Host Communications Program)
Memory registers Y:$0814, Y:08D4, etc., Y:$0817, Y:$08D7, etc.
I-variables I2, Ix11, Ix25;
On-line commands **?**, **??**, **HOME**, **{jog}^{constant}**.
Motion Program commands **{axis}{data}^{data}**

Ix13 Motor x Positive Software Position Limit

Range +/- 2^{35}

Units Encoder Counts

Default 0

Remarks Ix13 sets the position for Motor x which if exceeded in the positive direction causes a deceleration to a stop (controlled by Ix15) and allows no further positive position increments or positive output commands as long as the limit is exceeded. If this value is set to zero, there is no positive software limit (if you want 0 as a limit, use 1). This limit is automatically de-activated during homing search moves, until the home trigger is found. It is active during the post-trigger move.

Starting in firmware 1.15, bit 17 of Ix25 does not de-activate the software limits. Permanent de-activation is done by setting the value of the software limit to zero.

This limit is referenced to the most recent power-up zero position or homing move zero position. The physical position at which this limit occurs is not affected by axis offset commands (e.g. **`PSET, X=`**), although these commands will change the *reported* position value at which the limit occurs.

See Also Hardware Overtravel Limits, Software Overtravel Limits (Making Your Application Safe) I-variables Ix14, Ix15.

Ix14 Motor x Negative Software Position Limit

Range +/- 2^{35}

Units Encoder Counts

Default 0 (Disabled)

Remarks Ix14 sets the position for Motor x which if exceeded in the negative direction causes a deceleration to a stop (controlled by Ix15) and allows no further negative position increments or negative output commands as long as the limit is exceeded. If this value is set to zero, there is no negative software limit (if you want 0 as a limit, use -1). This limit is automatically de-activated during homing search moves, until the trigger is found. It is active during the post-trigger move.

Starting in firmware 1.15, bit 17 of Ix25 does not de-activate the software limits. Permanent de-activation is done by setting the value of the software limit to zero.

This limit is referenced to the most recent power-up zero position or homing move zero position. The physical position at which this limit occurs is not affected by axis offset commands (e.g. **`PSET, X=`**), although these commands will change the *reported* position value at which the limit occurs.

See Also Hardware Overtravel Limits, Software Overtravel Limits (Making Your Application Safe) I-variables Ix13, Ix15.

Ix15 Motor x Deceleration Rate on Position Limit or Abort

Range positive floating point

Units Counts/msec^2

Default 0.25

Remarks

WARNING:

Do not set this parameter to zero, or the motor will continue indefinitely after an abort or limit.

Ix15 sets the *rate* of deceleration that Motor x will use if it exceeds a hardware or software limit, or has its motion aborted by command (**A or <CONTROL-A)**. This value should usually be set to a value near the maximum physical capability of the motor. It is not a good idea to set this value past the capability of the motor, because doing so increases the likelihood of exceeding the following error limit, which stops the braking action, and could allow the axis to coast into a hard stop.

Example Suppose your motor had 125 encoder lines (500 counts) per millimeter, and you wished it to decelerate at 4000 mm/sec^2. You would set Ix15 to 4000 mm/sec^2 *500 cts/mm * sec^2/1,000,000 msec^2 = 2 cts/msec^2.

See Also On-line commands **A**, **<CONTROL-A>**
Hardware Overtravel Limits, Software Overtravel Limits (Making Your Application Safe)

Ix16 Motor x Maximum Permitted Motor Program

Range positive floating point

Units Counts/msec

Default 32.0

Remarks Ix16 sets a limit to the allowed velocity for **LINEAR** mode programmed moves for Motor x, provided I13 equals zero (no move segmentation). If a blended move command in a motion program requests a higher velocity of this motor, all motors in the coordinate system are slowed down proportionately so that Motor x will not exceed this parameter, yet the path will not be changed. This limit does not affect transition-point, circular, or splined moves. The calculation does not take into account any feedrate override value other than 100).

Note:

If PMAC's circular interpolation function is used at all, then I13 must be greater than zero, and Ix16 will not be active as a velocity limit.

This parameter also sets the speed of a programmed **RAPID** mode move for the motor, provided that variable I50 is set to 1 (if I50 is set to 0, jog speed parameter Ix22 is used instead). This happens regardless of the setting of I13.

With the Option 6L special lookahead firmware, Ix16 sets the limit for velocity of **LINEAR** and **CIRCLE** mode moves with segmentation active (I13 > 0).

See Also I-variables I13, I50, Ix17, Ix22
Velocity Limits (Making Your Application Safe)
LINEAR, **RAPID**-mode moves (Writing a Motion Program)

Ix17 Motor x Maximum Permitted Motor Program Acceleration

Range positive floating point

Units counts/msec^2

Default 0.5

Remarks Ix17 sets a limit to the allowed acceleration in **LINEAR**-mode blended programmed moves for Motor x, provided I13 equals zero (no move segmentation). If a **LINEAR** move command in a motion program requests a higher acceleration of this motor given its TA and TS time settings, the acceleration for all motors in the coordinate system is stretched out proportionately so that Motor x will not exceed this parameter, yet the path will not be changed.

Because PMAC cannot look ahead through an entire move sequence, it sometimes cannot anticipate enough to keep acceleration within this limit. Refer to LINEAR-mode trajectories in Writing a Motion Program.

It is possible to have this limit govern the acceleration for all LINEAR-mode moves by setting very low TA and TS times. Do not set both the TA and TS times to zero, or a division-by-zero error will occur in the move calculations, possibly causing erratic movement. The minimum acceleration time settings that should be used are TA1 with TS0.

Note:

When moves are broken into small pieces and blended together, this limit can affect the velocity, because it limits the calculated deceleration for each piece, even if that deceleration is never executed, because it blends into the next piece.

This limit does not affect **PVT**, **CIRCLE**, **RAPID**, or **SPLINE** moves. The calculation does not take into account any feedrate override value other than 100).

Note:

If PMAC's circular interpolation function is used at all, then I13 must be greater than zero, and Ix17 will not be active as an acceleration limit.

With the Option 6L special lookahead firmware, Ix17 sets the limit for acceleration of **LINEAR** and **CIRCLE** mode moves with segmentation active (I13 > 0). This mode of operation permits robust acceleration control by looking ahead far enough to ensure that these motor acceleration limits can always be obeyed.

Example Given axis definitions of **#1->10000X**, **#2->10000Y**, an Ix17 for each motor of 0.25, and the following motion program segment:

```
INC F10 TA200 TS0
X20
Y20
```

the rate of acceleration from the program at the corner for motor #2 (X) is ((0-10)units/sec * 10000 cts/unit * sec/1000msec) / 200 msec = -0.5 cts/msec^2. The acceleration of motor #2 (Y) is +0.5 cts/msec^2. Since this is twice the limit, the acceleration will be slowed so that it takes 400 msec.

With the same setup parameters, and the following program segment:

```
INC F10 TA200 TS0
X20 Y20
X-20 Y20
```

The rate of acceleration from the program at the corner for motor #1 (X) is ((-7.07-7.07)units/sec * 10000 cts/unit * sec/1000msec) / 200 msec = -0.707 cts/msec^2. The acceleration of motor #2 (Y) is 0.0. Since motor #1 exceeds its limit the acceleration time will be lengthened to 200 * 0.707/0.25 = 707 msec.

Note that in the second case, the acceleration time is made longer (the corner is made larger) for what is an identically shaped corner (90^o). In a contouring XY application, this parameter should not be relied upon to produce consistently sized corners.

See Also Acceleration Limits (Making Your Application Safe)
LINEAR-mode moves (Writing a Motion Program)
I-variables I13, I50, Ix16, Ix19, Ix22

Ix19 Motor x Maximum Permitted Motor Jog/Home Acceleration

Range positive floating point

Units counts/msec^2

Default 0.015625

Remarks Ix19 sets a limit to the commanded acceleration magnitude for jog and home moves, and for **RAPID**-mode programmed moves, of Motor x. If the acceleration times in force at the time (Ix20 and Ix21) request a higher rate of acceleration, this rate of acceleration will be used instead. The calculation does not take into account any feedrate override

Since jogging moves are usually not coordinated between motors, many people prefer to specify jog acceleration by rate, not time. To do this, simply set Ix20 and Ix21 low enough that the Ix19 limit is always used. Do not set both Ix20 and Ix21 to 0, or a division-by-zero error will result in the move calculations, possibly causing erratic operations. The minimum acceleration *time* settings that should be used are Ix20=1 and Ix21=0.

The default limit of 0.015625 counts/msec^2 is quite low and will probably limit acceleration to a lower value than is desired in most systems; most users will eventually raise this limit. This low default was used for safety reasons.

Example With Ix20 (accel time) at 100 msec, Ix21 (S-curve time) at 0, and Ix22 (jog speed) at 50 counts/msec, a jog command from stop would request an acceleration of (50 cts/msec) / 100 msec, or 0.5 cts/msec^2. If Ix19 were set to 0.25, the acceleration would be done in 200 msec, not 100 msec.

With the same parameters in force, an on-the-fly reversal from positive to negative jog would request an acceleration of (50-(-50) cts/msec) / 100 msec, or 1.0 cts/msec^2. The limit would extend this acceleration period by a factor of 4, to 400 msec.

See Also Jogging and Homing Moves (Basic Motor Moves)
RAPID-mode moves (Writing a Motion Program)
I-variables I50, Ix16, Ix20, Ix21, Ix22
On-line commands **HM, J+, J-, J=, J^, J:, J/**
Motion program commands **HOME**, **RAPID**

Motor Movement I-Variables

Ix20 Motor x Jog/Home Acceleration Time

Range 0 .. 8,388,607

Units msec

Default 0 (so Ix21 controls)

Remarks Ix20 establishes the time spent in acceleration in a jogging, homing, or programmed **RAPID**-mode move (starting, stopping, and changing speeds). However, if Ix21 (jog/home S-curve time) is greater than half this parameter, the total time spent in acceleration will be 2 times Ix21. Therefore, if Ix20 is set to 0, Ix21 alone controls the acceleration time in "pure" S-curve form. In addition, if the maximum acceleration rate set by these times exceeds what is permitted for the motor (Ix19), the time will be increased so that Ix19 is not exceeded.

Note:

Do not set both Ix20 and Ix21 to 0 simultaneously, even if you are relying on Ix19 to limit your acceleration, or a division-by-zero error will occur in the jog move calculations, possibly resulting in erratic motion.

A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time, command the jog, change the deceleration time, then command the jog move again (e.g. **J**), or at least the end of the jog (**J**).

See Also I-variables I50, Ix16, Ix19, Ix21, Ix22, Ix23, Ix87, Ix88
Jogging and Homing Moves (Basic Motor Moves)
RAPID-mode moves (Writing a Motion Program)
On-line commands **HM**, **J+**, **J-**, **J=**, **J^**, **J:**, **J/**
Motion program command **HOME**, **RAPID**

Ix21 Motor x Jog/Home S-Curve Time

Range 0 .. 8,388,607

Units msec

Default 50

Remarks Ix21 establishes the time spent in each half of the S for S-curve acceleration in a jogging, homing, or **RAPID**-mode move (starting, stopping, and changing speeds). If this parameter is more than half of Ix20, the total acceleration time will be two times Ix21, and the acceleration time will be "pure" S-curve (no constant acceleration portion). If the maximum acceleration rate set by Ix20 and Ix21 exceeds what is permitted for the motor (Ix19), the time will be increased so that Ix19 is not exceeded.

Note:

Do not set both Ix20 and Ix21 to 0 simultaneously, even if you are relying on Ix19 to limit your acceleration, or a division-by-zero error will occur in the jog move calculations, possibly resulting in erratic motion.

A change in this parameter will not take effect until the next move command. For instance, if you wanted a different deceleration time from acceleration time in a jog move, you would specify the acceleration time, command the jog, change the deceleration time, then command the jog move again (e.g. **J=**), or at least the end of the jog (**J/**).

See Also I-variables I50, Ix16, Ix19, Ix20, Ix22, Ix23, Ix87, Ix88
Jogging and Homing Moves (Basic Motor Moves)
RAPID-mode moves (Writing a Motion Program)
On-line commands **HM**, **J+**, **J-**, **J=**, **J^**, **J:**, **J/**
Motion program commands **HOME**, **RAPID**

Ix22 Motor x Jog Speed

Range positive floating point

Units Counts / msec

Default 32.0

Remarks Ix22 establishes the commanded speed of a jog move, or a programmed **RAPID**-mode move (if I50=0) for Motor x. Direction of the jog move is controlled by the jog command.

A change in this parameter will not take effect until the next move command. For instance, if you wanted to change the jog speed on the fly, you would start the jog move, change this parameter, then issue a new jog command.

See Also I-variables I50, Ix19-Ix21
Jogging Moves (Basic Motor Moves)
RAPID-Mode Moves (Writing a Motion Program)
On-line commands **J+**, **J-**, **J=**, **J^**, **J:**, **J/**
Program command **RAPID**

Ix23 Motor x Homing Speed and Direction

Range floating point

Units Counts / msec

Default 32.0

Remarks Ix23 establishes the commanded speed and direction of a homing-search move for Motor x. Changing the sign reverses the direction of the homing move – a negative value specifies a home search in the negative direction; a positive value specifies the positive direction.

See Also Homing Moves (Basic Motor Moves)
I-variables Ix19, Ix20 Ix21, Ix22, Ix25, Ix26
Encoder/Flag I-Variables 2 and 3
On-line command **HM**
Motion program command **HOME**.

Ix24 (Reserved for Future Use)

Ix25 Motor x Limit/Home Flag

Range Extended legal PMAC X addresses

Units Extended legal PMAC X addresses

Default

Variable	PMAC(1)	PMAC2	PMAC2 Ultralite
I125	$C000	$C000	$40F70
I225	$C004	$C008	$40F71
I325	$C008	$C010	$40F74
I425	$C00C	$C018	$40F75
I525	$C010	$C020	$40F78
I625	$C014	$C028	$40F79
I725	$C018	$C030	$40F7C
I825	$C01C	$C038	$40F7D

Remarks This parameter tells PMAC what set of flags it will look to for Motor x's overtravel limit switches, home flag, amplifier-fault flag, amplifier-enable output, and index channel. Typically, these are the flags associated with an encoder input; specifically, those of the position feedback encoder for the motor. If dual-loop feedback is used (Ix03 and Ix04 are different) Ix25 should be set to match the position-loop encoder, not the velocity-loop.

Note:

To use PMAC's Hardware Position Capture for homing search moves, the channel number of the flags specified by Ix25 must match the channel number of the encoder specified by Ix03 for position-loop feedback.

The addresses for the flag sets in PMAC(1) and PMAC2 systems are given in the tables below. Channel3s 9 – 16 are present on ACC-24 axis-expansion boards, not on the PMACs themselves.

PMAC(1)

Channel	Address	Channel	Address
1	$C000	9	$C020
2	$C004	10	$C024
3	$C008	11	$C028
4	$C00C	12	$C02C
5	$C010	13	$C030
6	$C014	14	$C034
7	$C018	15	$C038
8	$C01C	16	$C03C

PMAC2

Channel	Address	Channel	Address
1	$C000	9	$C040
2	$C008	10	$C048
3	$C010	11	$C050
4	$C018	12	$C058
5	$C020	13	$C060
6	$C028	14	$C068
7	$C030	15	$C070
8	$C038	16	$C078

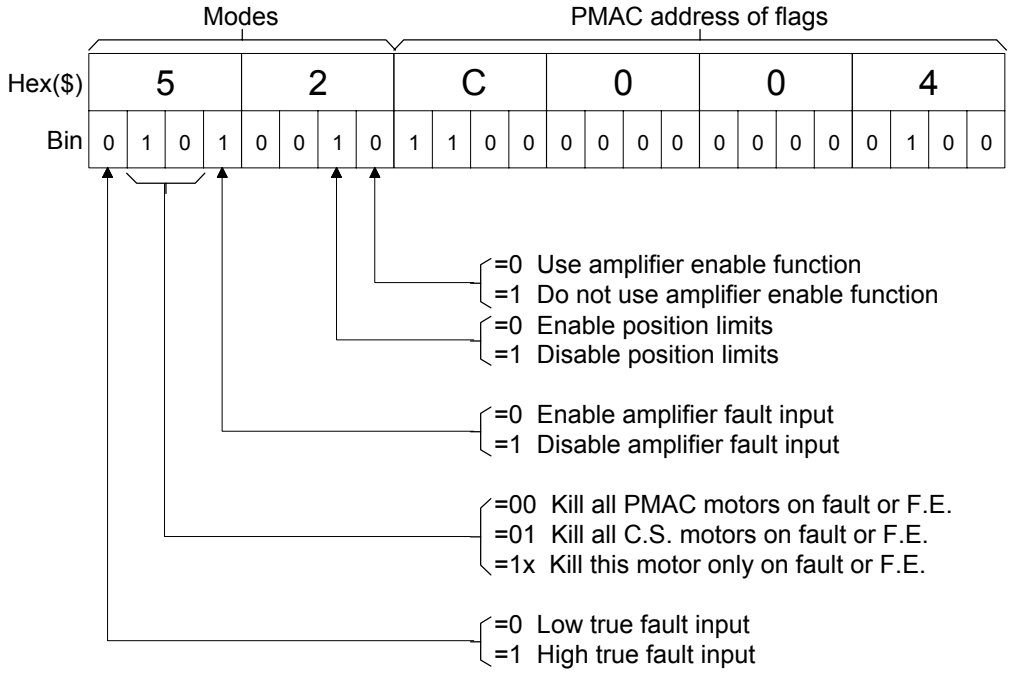
The overtravel-limit inputs specified by this parameter must be held low in order for Motor x to be able to command movement. The polarity of the amplifier-fault input is determined by a high-order bit of this parameter (see below). The polarity of the home-flag input is determined by the Encoder/Flag I-Variables 2 and 3 for the specified encoder. The polarity of the amplifier-enable output is determined by Jumper E17.

Extended Addressing: The source address of the flag information occupies bits 0 to 15 of Ix25 (range $0000 to $FFFF, or 0 to 65535). If this is all that is specified – that is, all higher bits are zero – then all of the flags are used, and used in the "normal" mode (low-true FAULT, disabling all motors). If higher bits are set to one, some of the flags are not used, or used in an alternate manner, as documented below.

Note:

In the extended versions, it is easier to specify this parameter in hexadecimal form. With I9 at 2 or 3, the value of this variable will be reported back to the host in hexadecimal form.

Ix25 - Motor x Flag Address and Modes



Amplifier Enable Use Bit: With bit 16 equal to zero – the normal case – the AENAn/DIRn output is used as an amplifier-enable line: off when the motor is killed, on when it is enabled. Voltage polarity is determined by jumper(s) E17.

If bit 16 (value $10000, or 65536) is set to one (e.g. I125=$1C000), this output is not used as an amplifier-enable line. This permits use of the line as a direction bit for applications requiring magnitude-and direction outputs, such as driving steppers through voltage-to-frequency converters. (Setting bit 16 of Ix02 to 1 enables use of this output as a direction bit.) General-purpose use of this output is also possible by assigning an M-variable to it.

Overtravel Limit Use Bit: With bit 17 equal to zero – the normal case – the +/-LIMn inputs must be held low to permit commanded motion in the appropriate direction. If there are not actual (normally closed or normally conducting) limit switches, the inputs must be hardwired to ground.

Note:

The direction sense of the limit inputs is the opposite of what many people consider intuitive. That is, the +LIMn input, when taken high (opened), stops commanded motion in the negative direction; the -LIMn input, when taken high, stops commanded motion in the positive direction. It is important to confirm the direction sense of your limit inputs in actual operation.

If bit 17 (value $20000, or 131072) is set to one (e.g. I125=$2C000), Motor x does not use these inputs as overtravel limits. This can be done temporarily, as when using a limit as a homing flag. If the limit function is not used at all, these inputs can be used as general-purpose inputs by assigning M-variables to them.

Starting in firmware 1.15, bit 17 of Ix25 does not effect the software overtravel limits. Activation of the software overtravel limits is done by setting the value of Ix13 and or Ix14 to a non-zero value. De-activation is done by setting their values to zero.

MACRO Flag Bit: If bit 18 of Ix25 is 0, the flag set is wired directly into the PMAC controller. If bit 18 (value $40000, or 262,144) is 1 (e.g. I125=$4070), PMAC looks for these flags to come through the MAXCRO ring.

Amplifier Fault Use Bit: If bit 20 of Ix25 is 0, the amplifier-fault input function through the FAULTn input is enabled. If bit 20 (value $100000, or 1, 048,576) is 1 (e.g. 1125=$10C000), this function is disabled. General-purpose use of this input is then possible by assigning an M-variable to the input.

Action-on-Fault Bits: Bits 21 (value $200000, or 2,097,152) and 22 (value $400000, or 4,194,344) of Ix25 control what action is taken on an amplifier fault for the motor, or on exceeding the fatal following err limit (Ix11) for the motor:

Bit 22	Bit 21	Function
Bit 22=0	Bit 21=0:	Kill all PMAC motors
Bit 22=0	Bit 21=1:	Kill all motors in same coordinate system
Bit 22=1	Bit 21=0:	Kill only this motor
Bit 22=1	Bit 21=1:	Kill only this motor

Regardless of the setting of these bits, a program running in the coordinate system of the offending motor will be halted on an amplifier fault of the exceeding of a fatal following error limit.

Amplifier-Fault Polarity Bit: Bit 23 (value 8,388,608) of Ix25 controls the polarity of the amplifier fault input. A zero in this bit means a low-true input (low means fault); a one means high-true (high means fault). The input is pulled high internally, so if no line is attached to the input, and bit 20 of Ix25 is zero (enabling the fault function), bit 23 of Ix25 must be zero to permit operation of the motor.

First Hex Digit: In the hexadecimal form, bits 20 to 23 combine to form a single hexadecimal digit.

For reference, the possible values and their meanings are:

Hex Digit	Function
$0:	Low-true amp fault enabled; all motors killed on fault or excess following error (default)
$1:	Amp fault disabled; all motors killed on excess following error
$2:	Low-true amp fault enabled: coordinate system motors killed on fault or excess following error
$3	Amp fault disabled; coordinate system motors killed on excess following error
$4	Low-true amp fault enabled; only this motor k8illed on fault or excess following error
$5:	Amp fault disabled; only this motor killed on excess following error
$6:	Low-true amp fault enabled; only this motor killed on fault or excess following error
$7:	Amp fault disabled; only this motor killed on excess following error
$8	High-true amp fault enabled; all motors killed on fault or excess following error (default)
$9:	Amp fault disabled; all motors killed on excess following error
$A:	High-true amp fault enabled; coordinate system motors killed on excess following error
$B:	Amp fault disabled; coordinate system motors killed on excess following error
$C:	High-true amp fault enabled; only this motor killed on fault or excess following error
$D:	Amp fault disabled; only this motor killed on excess following error
$E:	High-true amp fault enabled; only this motor killed on fault or excess following error
$F:	Amp fault disabled; only this motor killed on excess following error

Example

1. Motor 1 using flags 1 with amp-enable output, and low-true amp fault disabling all motors: **`I125=$00C000`** or **`I125=$C000`**
2. Motor 1 using flags 1 with direction output, and low-true amp fault disabling all motors:
   `I125=$01C000`
3. Motor 1 using flags 1 with amp-enable output, and low-true amp fault disabling only coordinate system motors: **`I125=20C000`**
4. Motor 1 using flags 1 with direction output, and amp-fault disabled, with excess F.E. disabling off C.S motors: **`I125=$31C000`**
5. Motor 1 using flags 5 with amp-enable output, and high-true amp fault disabling only this motor: **`I125=$C0C010`**

See Also

Selecting the Flag Register (Setting up a Motor)
Homing Moves (Basic Motor Moves)
I-variables Ix02, Ix03 Ix11
Encoder/Flag I-Variables 2 and 3
Jumper(s) E17; JMACH connector flag I/O pins

Ix26 Motor x Home Offset

Range -8,388,608 .. 8,388,607

Units 1/16 Count

Default 0

Remarks Ix26 specifies the relative distance of the Motor x zero position to either the trigger position of a homing search move, or the zero position of an absolute sensor.

If Ix10 is set to 0, PMAC presumes the motor uses an incremental sensor and sets motor position to 0 on power-up/reset. A homing search move is then required to establish the true machine zero position.

In the homing search move, PMAC moves the motor until it sees a pre-defined trigger condition, either an input trigger defined by Encoder I-variables 2 and 3 for the servo interface channel addressed by Ix25, or the exceeding of the warning following error as set by Ix12.

In the post-trigger part of the homing search move, the motor will stop a distance of Ix26 from the position at which it found the trigger, and call this commanded location as motor position zero.

This permits the motor zero position to be at a different location from the home trigger position, particularly useful when using an overtravel limit as a home flag (offsetting out of the limit before re-enabling the limit input as a limit). If large enough (greater than 1/2 times home speed times accel time) it permits a homing move without any reversal of direction.

If Ix10 is greater than 0, PMAC reads the sensor specified by Ix10 for the motor's absolute position. In this case, it subtracts Ix26 from the sensor position to calculate absolute motor position. This is especially desirable if the zero position of the sensor is outside the region of travel, as it is for an MLDT.

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts between the trigger position and the home zero position.

Example If you wish your motor zero position to be 500 counts in the negative direction from the home trigger position, you would set Ix26 to -500 * 16 = -8000.

See Also Homing Moves (Basic Motor Moves)
Absolute Power-Up Position (Setting Up a Motor)
I-variables Ix10, Ix23, Ix25
Encoder I-Variables 2 and 3
On-line command **HM**
Program command **HOME**

Ix27 Motor x Position Rollover Range

Range $+/-2^{35}$

Units Counts

Default 0

Remarks This parameter permits special position rollover modes on a PMAC rotary axis assigned to Motor x by telling PMAC how many encoder counts are in one revolution of the rotary axis. This lets PMAC compute the rollover function properly. If Ix27 is set to a non-zero value, and Motor x is assigned to a rotary axis (A, B, or C), rollover is active.

If Ix27 is set to a value greater than zero, for a programmed axis move in Absolute (**ABS**) mode, the motor will take the shortest path around the circular range defined by Ix27 to get to the destination point. No absolute move will be greater than half a revolution in this mode.

If Ix27 is set to a negative number, an alternate rollover mode for the rotary axis assigned to the motor is activated that uses the sign of the commanded destination in absolute mode to specify the direction of motion to that destination. In this mode, all absolute-mode moves are less than one revolution (with the size of the revolution specified by the magnitude of Ix27), but can be greater than one-half revolution.

The sign of the commanded absolute destination in this mode is also part of the destination value. So a command of **A-90** in this mode is a command to move to –90 degrees (= +270 degrees) in the negative direction. For commands to move in the positive direction, the + sign is not required, but it is permitted (e.g. to command a move to 90 degrees in the positive direction, either **A90** or **A+90** can be used).

PMAC cannot store the difference between a +0 and a –0 destination command, so a command with a tiny non-zero magnitude must be used (e.g. **A+0.0000001** and **A-0.0000001**). This increment can be small enough not to have any effect on the final destination.

If using commands from a similar mode in which only the magnitude, and not the sign, of the value specifies the destination position, then the destination values for negative-direction moves must be modified so that the magnitude is 360 degrees minus the magnitude in the other mode. For example, if the command were **C-120**, specifying a move to (+)120 degrees in the negative direction, the command would have to be modified for PMAC to **C-240**, which specifies a move to –240 degrees (= +120 degrees) in the negative direction. Commands for positive-direction moves do not have to be modified.

Axis moves in Incremental (**INC**) mode are not affected by rollover. When Ix27 is set to 0, there is no rollover. Rollover should not be attempted for axes other than A, B, or C. Jog moves are not affected by rollover. Reported motor position is not affected by rollover. (To obtain motor position information "rolled over" to within one motor revolution, use the modulo (remainder) operator, either in PMAC or in the host computer: e.g. **P4=(M462/(I408*32))%I427)**.

Example

Motor #4 drives a rotary table with 36,000 counts per revolution. It is defined to the A-axis with **#4->100A** (A is in units of degrees). I427 is set to 36000. With motor #4 at zero counts (A-axis at zero degrees), an **A270** move in a program is executed in Absolute mode. Instead of moving the motor from 0 to 27,000 counts, which it would have done with I427=0, PMAC moves the motor from 0 to -9,000 counts, or -90 degrees, which is equivalent to +270 degrees on the rotary table.

Motor #5 drives a positioning spindle with an 8192-line-per-rev (32,768-count-per-rev) encoder on the motor and a 10-to-1 gear reduction to the load. It is defined to the C-axis with **#5->910.222222222C** (C is in units of degrees). I527 is set to 327,680. An absolute-mode move of **C-355** is commanded. PMAC moves Motor 5 in the negative direction less than one revolution to +5 degrees (= -355 degrees).

See Also

On-line commands **INC**, **ABS**
Program commands **INC**, **ABS, A{data}, B{data}, C{data}**
Axis Types (Setting Up a Coordinate System)

Ix28 Motor x In-position Band

Range 0 .. 8,388,607

Units 1/16 Count

Default 160 (10 counts)

Remarks Ix28 determines the magnitude of the maximum following error at which Motor x will be considered "in position" when not performing a move. Several things happen when the motor is "in-position". First, a status bit in the motor status word is set. Second, if all other motors in the same coordinate system are also "in-position", a status bit in the coordinate system status word is set. Third, for the hardware-selected (FPD0/-FPD3/) coordinate system – if I2=0 – or for the software addressed (&n) coordinate system – if I2=1 – outputs to the control panel port and to the interrupt controller are set.

Technically, five conditions must be met for a motor to be considered "in-position":

1. The motor must be in closed-loop control
2. The desired velocity must be zero;
3. The magnitude of the following error must be less than this parameter;
4. The move timer must not be active;
5. The above four conditions must all be true for (I7+1) consecutive scans

The move timer is active during any programmed or non-programmed move, including **DWELL**s and **DELAY**s in a program – if you wish this bit to come true during a program, you must do an indefinite wait between some moves by keeping the program trapped in a **WHILE** loop that has no moves or **DWELL**s. More sophisticated in-position functions (for instance, ones that require several consecutive scans within the band) can be implemented using PLC programs. See the program examples section.

Note:

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts in the in-position band.

Example

```
M140->Y:$0814,0                 ; Motor 1 in-position bit
...
WHILE (M140=0) WAIT             ; Delay indefinitely until in-position is true
M1=1                            ; Set output once in-position
```

See Also

Control Panel Port (Connecting PMAC to the Machine)
Using Interrupts (Writing a Host Communications Program)
I-variable I7
On-line commands **?**, **??**
Suggested M-variable definitions Mx40, Mx87
Memory Registers Y:$0814, Y:$08D4, etc., Y:$0817, Y:$08D7, etc.
JPAN connector

Ix29 Motor x Output/First Phase Offset

Range -32,768 .. 32,767

Units 16-bit DAC/ADC bit equivalent

Default 0

Remarks Ix29 serves as an output or feedback offset for Motor x; its exact use depends on the mode of operation as described below. In any of the modes, it effectively serves as the digital equivalent of an offset pot.

Mode 1*:* When PMAC is not commutating Motor x (Ix01 = 0), Ix29 serves as the offset for the single command output value, usually a DAC command. Ix29 is added to the output command value before it is written to the command output register.

Mode 2: When PMAC (PMAC(1) only) is not commutating Motor x (Ix01 = 0) but is in sign-and-magnitude output mode (Ix02 bit 16 = 1), Ix29 is the offset of the command output value before the absolute value is taken (Ix79 is the offset after the absolute value is taken). Ix29 is typically left at zero in this mode, because it cannot compensate for real circuitry offsets.

Mode 3: When PMAC is commutating Motor x (Ix01 Bit 0 = 1) but not closing the current loop (Ix82 = 0), Ix29 serves as the offset for the first of two phase command output values (Phase A), for the address specified by Ix02; Ix79 serves the same purpose for the second phase (Phase B). Ix29 is added to the output command value before it is written to the command output register.

When commutating from a PMAC(1), Phase A is output on the *higher*-numbered of the two DACs (e.g. DAC2), Phase B on the *lower*-numbered (e.g. DAC1). When commutating from a PMAC2, Phase A is output on the A-channel DAC (e.g. DAC1A), Phase B on the B-channel DAC (e.g. DAC1B).

As an output command offset, Ix29 is always in units of a 16-bit register, even if the actual output device is of a different resolution. For example, if a value of 60 had to be written into an 18-bit DAC to create a true zero command, this would be equivalent to a value of 60/4=15 in a 16-bit DAC, so Ix29 would be set to 15 to cancel the offset.

Mode 4: When PMAC (PMAC2 only) is commutating (Ix01 Bit 0 = 1) and closing the current loop for Motor x (Ix82 > 0), Ix29 serves as an offset that is added to the phase current reading from the ADC for the first phase (Phase A), at the address specified by Ix82 minus 1. Ix79 performs the same function for the second phase. The sum of the ADC reading and Ix29 is used in the digital current loop algorithms.

As an input feedback offset, Ix29 is always in units of a 16-bit ADC, even if the actual ADC is of a different resolution. For example, if a 12-bit ADC reported a value of -5 when no current was flowing in the phase, this would be equivalent to a value of -5*16=-80 in a 16-bit ADC, so Ix29 would be set to 80 to compensate for this offset.

See Also Setting Up PMAC Commutation

I-variables Ix01, Ix02, Ix79.

Servo Control I-Variables

Ix30 – Ix58 Motor x Extended Servo Algorithm Gains {Option 6 firmware only}

Range 0.0 – 0.999999

Units none

Default 0.0

Remarks When the Option 6 Extended Servo Algorithm (ESA) special firmware is ordered, variables Ix30 through Ix58 for each Motor x have different meanings from those in the standard firmware with the PID servo filter. The following table shows the meanings of these variables for the ESA algorithm. Please refer to the block diagram of the ESA in the User's Manual to understand the function of each of these variables.

I-Variable	Gain Name	Range	I-Variable	Gain Name	Range
Ix30	s0	-1.0≤Var<+1.0	Ix45	TS	-2^{23}≤Var<2^{23}
Ix31	s1	-1.0≤Var<+1.0	Ix46	L1	-1.0≤Var<+1.0
Ix32	f0	-1.0≤Var<+1.0	Ix47	L2	-1.0≤Var<+1.0
Ix33	f1	-1.0≤Var<+1.0	Ix48	L3	-1.0≤Var<+1.0
Ix34	h0	-1.0≤Var<+1.0	Ix49	k0	-1.0≤Var<+1.0
Ix35	h1	-1.0≤Var<+1.0	Ix50	k1	-1.0≤Var<+1.0
Ix36	r1	-1.0≤Var<+1.0	Ix51	k2	-1.0≤Var<+1.0
Ix37	r2	-1.0≤Var<+1.0	Ix52	k3	-1.0≤Var<+1.0
Ix38	r3	-1.0≤Var<+1.0	Ix53	KS	-2^{23}≤Var<2^{23}
Ix39	r4	-1.0≤Var<+1.0	Ix54	d1	-1.0≤Var<+1.0
Ix40	t0	-1.0≤Var<+1.0	Ix55	d2	-1.0≤Var<+1.0
Ix41	t1	-1.0≤Var<+1.0	Ix56	g0	-1.0≤Var<+1.0
Ix42	t2	-1.0≤Var<+1.0	Ix57	g1	-1.0≤Var<+1.0
Ix43	t3	-1.0≤Var<+1.0	Ix58	GS	-2^{23}≤Var<2^{23}
Ix44	t4	-1.0≤Var<+1.0	(Ix68)	Kfff	0≤Var<2^{16}

Variables Ix59, Ix63, Ix64, Ix65, Ix66, and Ix67 for the standard PID algorithm have no function for the Option 6 ESA.

Ix30 Motor x PID Proportional Gain

Range -8,388,608 .. 8,388,607

Units (Ix08/2^{19}) DAC bits/Encoder count

Default 2000

Remarks

WARNING:

Changing the sign of Ix30 on a motor that has been closing a stable servo loop will cause an unstable servo loop, leading to a probable runaway condition.

Ix30 provides a control output proportional to the position error (commanded position minus actual position) of Motor x. It acts effectively as an electronic spring. The higher Ix30 is, the stiffer the "spring" is. Too low a value will result in sluggish performance. Too high a value can cause a "buzz" from constant over-reaction to errors.

If Ix30 is set to a negative value, this has the effect of inverting the command output polarity for motors not commutated by PMAC, when compared to a positive value of the same magnitude. This can eliminate the need to exchange wires to get the desired polarity. On a motor that is commutated by PMAC, changing the sign of Ix30 has the effect of changing the commutation phase angle by 180°. Negative values of Ix30 cannot be used with the auto-tuning programs in the PMAC Executive program.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

Note:

The default value of 2000 for this parameter is exceedingly weak for most systems (all but the highest resolution velocity-loop systems), causing sluggish motion and/or following error failure. Most users will immediately want to raise this parameter significantly even before starting serious tuning.

If the servo update time is changed, Ix30 will have the same effect for the same numerical value. However, smaller update times (faster update rates) should permit higher values of Ix30 (stiffer systems) without instability problems.

See Also PID Servo Filter (Closing the Servo Loop)
I-variables Ix31-Ix39
Tuning Instructions (PMAC Executive Program manual)

Ix31 Motor x PID Derivative Gain

Range -8,388,608 .. 8,388,607

Units (Ix30*Ix09)/2^{26} DAC bits/(Counts/cycle)

Default 1280

Remarks Ix31 subtracts an amount from the control output proportional to the measured velocity of Motor x. It acts effectively as an electronic damper. The higher Ix31 is, the heavier the damping effect is.

If the motor is driving a properly tuned tachometer-loop (velocity) amplifier, the amplifier will provide sufficient damping, and Ix31 should be set to zero. If the motor is driving a current-loop (torque) amplifier, or if PMAC is commutating the motor, the amplifier will provide no damping, and Ix31 must be greater than zero to provide damping for stability.

On a typical system with a current-loop amplifier and PMAC's default servo update time (~440 μsec), an Ix31 value of 2000 to 3000 will provide a critically damped step response.

If the servo update time is changed, Ix31 must be changed proportionately in the opposite direction to keep the same damping effect. For instance, if the servo update time is cut in half, from 440 μsec to 220 μsec, Ix31 must be doubled to keep the same effect.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

See Also I-variables Ix30, Ix32-Ix39
PID Servo Filter (Closing the Servo Loop)
Tuning Instructions (PMAC Executive Program manual)

Ix32 Motor x PID Velocity Feedforward Gain

Range 0 .. 8,388,607

Units (Ix30*Ix08)/2^{26} DAC bits/(counts/cycle)

Default 1280

Remarks Ix32 adds an amount to the control output proportional to the desired velocity of Motor x. It is intended to reduce tracking error due to the damping introduced by Ix31, analog tachometer feedback, or physical damping effects.

If the motor is driving a current-loop (torque) amplifier, Ix32 will usually be equal to (or slightly greater than) Ix31 to minimize tracking error. If the motor is driving a tachometer-loop (velocity) amplifier, Ix32 will typically be substantially greater than Ix31 to minimize tracking error.

If the servo update time is changed, Ix32 must be changed proportionately in the opposite direction to keep the same effect. For instance, if the servo update time is cut in half, from 440 μsec to 220 μsec, Ix32 must be doubled to keep the same effect.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

See Also PID Servo Filter (Closing the Servo Loop)
I-variables Ix30-Ix31, Ix32-Ix39
Tuning Instructions (PMAC Executive Program manual)

Ix33 Motor x PID Integral Gain

Range 0 .. 8,388,607

Units (Ix30*Ix08)/2^{42} DAC bits/(counts*cycles)

Default 0

Remarks Ix33 adds an amount to the control output proportional to the time integral of the position error for Motor x. The magnitude of this integrated error is limited by Ix63. With Ix63 at a value of zero, the contribution of the integrator to the output is zero, regardless of the value of Ix33.

No further errors are added to the integrator if the output saturates (if output equals Ix69), and, if Ix34=1, when a move is being commanded (when desired velocity is not zero). In both of these cases, the contribution of the integrator to the output remains constant.

If the servo update time is changed, Ix33 must be changed proportionately in the same direction to keep the same effect. For instance, if the servo update time is cut in half, from 440 μsec to 220 μsec, Ix33 must be cut in half to keep the same effect.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

See Also PID Servo Filter (Closing the Servo Loop)
I-variables Ix30-Ix32, Ix34-Ix39, Ix63, Ix69
Tuning Instructions (PMAC Executive Program manual)

Ix34 Motor x PID Integration Mode

Range 0 .. 1

Units none

Default 1

Remarks Ix34 controls when the position-error integrator is turned on. If it is 1, position error integration is is the input to the integrator that is turned off during a commanded move, which means performed only when PMAC is not commanding a move (when desired velocity is zero). If it is 0, position error integration is performed all the time.

If Ix34 is 1, it the output control effort of the integrator is kept constant during this period (but is generally not zero). This same action takes place whenever the total control output saturates at the Ix69 value.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. When performing the feedforward tuning part of that utility, it is important to set Ix34 to 1 so the dynamic behavior of the system may be observed without integrator action. Ix34 may be changed on the fly at any time to create types of adaptive control.

See Also PID Servo Filter (Closing the Servo Loop)
I-variables Ix30-Ix33, Ix35-Ix39, Ix63, Ix69
Tuning Instructions (PMAC Executive Program manual)

Ix35 Motor x PID Acceleration Feedforward Gain

Range 0 .. 8,388,607

Units $(Ix30*Ix08)/2^{26}$ DAC bits/(counts/cycle^2)

Default 0

Remarks Ix35 adds an amount to the control output proportional to the desired acceleration for Motor x. It is intended to reduce tracking error due to inertial lag.

If the servo update time is changed, Ix35 must be changed by the inverse square to keep the same effect. For instance, if the servo update time is cut in half, from 440 μsec to 220 μsec, Ix35 must be quadrupled to keep the same effect.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

See Also PID Servo Filter (Closing the Servo Loop)
I-variables Ix30-Ix34, Ix36-Ix39
Tuning Instructions (PMAC Executive Program manual)

Ix36 Motor x PID Notch Filter Coefficient N1

Range -2.0 .. +2.0

Units none (actual z-transform coefficient)

Default 0

Remarks Ix36, along with Ix37 – Ix39, is part of the notch filter for Motor x, whose purpose is to damp out a resonant mode in the system. This parameter can be set according to instructions in the Servo Loop Features section of the manual.

The notch filter parameters Ix36-Ix39 are 24-bit variables, with 1 sign bit, 1 integer bit, and 22 fractional bits, providing a range of -2.0 to +2.0.

The equation for the notch filter is:

$$F(z) = \frac{1 + N1z^{-1} + N2z^{-2}}{1 + D1z^{-1} + D2z^{-2}}$$

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

See Also Notch Filter (Closing the Servo Loop)
I-variables Ix30-Ix35, Ix37-Ix39
Tuning Instructions (PMAC Executive Program manual)

Ix37 Motor x PID Notch Filter Coefficient N2

Range -2.0 .. +2.0

Units none (actual z-transform coefficient)

Default 0

Remarks Ix37 is part of the notch filter for Motor x. See Ix36 and the Servo Loop Features section of the manual for details.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

See Also Notch Filter (Closing the Servo Loop)
I-variables Ix30-Ix36, Ix38-Ix39
Tuning Instructions (PMAC Executive Program manual)

Ix38 Motor x PID Notch Filter Coefficient D1

Range -2.0 .. +2.0

Units none (actual z-transform coefficient)

Default 0

Remarks Ix38 is part of the notch filter for Motor x. See Ix36 and the Servo Loop Features section of the manual for details.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

See Also Notch Filter (Closing the Servo Loop)
I-variables Ix30-Ix37, Ix39
Tuning Instructions (PMAC Executive Program manual)

Ix39 Motor x PID Notch Filter Coefficient D2

Range -2.0 .. +2.0

Units none (actual z-transform coefficient)

Default 0

Example Ix39 is part of the notch filter for Motor x. See Ix36 and the Servo Loop Features section of the manual for details.

This parameter is usually set initially using the Tuning utility in the PMAC Executive Program. It may be changed on the fly at any time to create types of adaptive control.

See Also Notch Filter (Closing the Servo Loop)
I-variables Ix30-Ix38
Tuning Instructions (PMAC Executive Program manual)

Ix40 - Ix56 Motor x Extended Servo Algorithm I-Variables

(These variables are used only with the Option 6 Extended Servo Algorithm. Refer to the manual for the Extended Servo Algorithm and the ACC-25 Servo Evaluation Program for details.)

Ix40 Motor x Net Desired Position Filter Gain {Option 6L Firmware Only}

Range 0.0 – 0.999999

Units none

Default 0.0

Remarks Ix40 permits the introduction of a first-order low-pass filter on the net desired position for Motor x. This can be useful to smooth motion that comes from a "rough" source, such as master following from a noisy sensor, or quantization error in very closely spaced programmed points that are commonly found in lookahead applications.

If Ix40 is set to its default value of .0, this filter function is disabled. If Ix40 is set to any value greater than 0.0, the filter is enabled.

Ix40 can be expressed in terms of the filter time constant by the following equation:

$$Ix40 = \frac{T_f}{T_S + T_f}$$

where T_f is the filter time constant, and T_s is the servo update time.

The filter time constant can be expressed in terms of Ix40 by the following equation:

$$Tf = \frac{Ix40 * T_S}{1 - Ix40}$$

Filter time constants can range from a fraction of a servo cycle (when Ix40 ~ 0) to infinite (when Ix40 ~ 1). As with any low-pass filter, there is a fundamental trade-off between smoothness and delay. Generally when the filter is used, filter time constants of a few milliseconds are set. In an application where multiple motors are executing a path, the same time constant should be sued for all of the motors.

Ix40 is available only with the special Lookahead option. If the Extended Servo Algorithm option is selected along with the Lookahead option, this Ix40 filter is not available. (In the ESA, Ix40 is used for another purpose.)

Example To set a filter time constant of 2 msec on a system with the default servo update time of 442 μsec, Ix40 can be computed as:

$$Ix40 = \frac{2}{0.442 + 2} = 0.819$$

Motor Servo Loop Modifiers

These I-variables modify the action of the basic PID servo algorithm. They are not available with the Option 6 Extended Servo Algorithm firmware.

Ix57 Motor x Continuous Current Limit

Range 0 .. 32,767

Units 16-bit DAC/ADC bit equivalent

Default 0

Remarks Ix57 sets the maximum continuous current limit for PMAC's I^2T integrated current limiting function, when that function is active (Ix58 must be greater than 0 for I^2T to be active). If PMAC is closing a digital current loop for the motor, it uses actual current measurements for this function; otherwise it uses commanded current values. If the actual or commanded current level from PMAC is above Ix57 for a significant period of time, as set by Ix58, PMAC will trip this motor on an I^2T amplifier fault condition.

Ix57 is in units of a 16-bit DAC or ADC (maximum possible value of 32,767), even if the actual output or input device has a different resolution. Typically Ix57 will be set to between 1/3 and 1/2 of the Ix69 (instantaneous) output limit. Consult your amplifier and motor documentation for their specifications on instantaneous and continuous current limits.

Technically, Ix57 is the continuous limit of the vector sum of the quadrature and direct currents. The quadrature (torque-producing) current is the output of the position/velocity-loop servo. The direct (magnetization) current is set by Ix77.

In sine-wave output mode (Ix01 = 1, Ix82 = 0), amplifier gains are typically given in amperes of phase current per volt of PMAC output, but motor and amplifier limits are typically given in RMS amperage values. In this case, it is important to realize that peak phase current values are √2 (1.414) times greater than the RMS values.

In direct-PWM mode (Ix01 = 1, Ix82 > 0) of 3-phase motors (Ix72 = 85 or 171), the corresponding top values of the sinusoidal phase-current ADC readings will be $1/\cos(30^o)$, or 1.15, times greater than the vector sum of quadrature and direct current. Therefore, once you have established the top values you want to see in the A/D converters your phase currents on a continuous basis, this value should be multiplied by $\cos(30^o)$, or 0.866, to get your value for Ix57. Remember that if current limits are given as RMS values, you should multiply these by √2 (1.414) to get peak phase current values.

Example

1. PMAC Motor 1 is driving a torque-mode DC brush-motor amplifier that has a gain of 3 amps/volt with a single analog output voltage. The amplifier has a continuous current rating of 10 amps; the motor has a continuous current rating of 12 amps.
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 30 amps.
   - The amplifier has the lower continuous current rating, so we use its limit of 10 amps.
   - I157 is set to 32,768 * 10 / 30 = 10,589.
2. Motor 3 is driving a self-commutating brushless-motor amplifier in current (torque) mode with a single analog output. The amplifier has a gain of 5 amps(RMS)/volt and an continuous current limit of 20 amps (RMS). The motor has an continuous current limit of 25 amps (RMS).
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 50 amps (RMS).
   - The amplifier has the lower continuous current rating, so we use its limit of 20 amps (RMS).

- I357 is set to 32,768 * 20/50 = 13,107.

3. PMAC Motor 4 is driving a sine-wave mode amplifier that has a gain for each phase input of 5 amps/volt. The amplifier has a continuous rating of 20 amps (RMS); the motor has a continuous rating of 22 amps (RMS).
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 50 amps peak in a phase.
   - The amplifier has the lower continuous current rating, so we use its limit of 20 amps (RMS).
   - 20 amps (RMS) corresponds to peak phase currents of 20*1.414 = 28.28 amps.
   - I457 is set to 32,768 * 28.28 / 50 = 18,534.
4. PMAC Motor 6 is driving a direct-PWM power block amplifier for an AC motor. The A/D converters in the amplifier are scaled so that a maximum reading corresponds to 50 amps of current in the phase. The amplifier has a continuous current rating of 20 amps (RMS), and the motor has a continuous rating of 15 amps (RMS).
   - PMAC's maximum ADC phase reading of 32,768 corresponds to 50 amps.
   - The motor has the lower continuous current rating, so we use its limit of 15 amps (RMS).
   - 15 amps (RMS) corresponds to peak phase currents of 15*1.414 = 21.21 amps.
   - 21.21 amps corresponds to an ADC reading of 32,768 * 21.21/50 = 13,900.
   - I657 should be set to 13,900 * 0.866 = 12,037.

See Also Integrated Current Protection (Making Your Application Safe)
I-Variables Ix58, Ix69

Ix58 Motor x Integrated Current Limit

Range 0 .. 8,388,607

Units 2^{30} (DAC bits)^2 • servo cycles
{bits of a 16-bit DAC}

Default 0

Remarks Ix58 sets the maximum integrated current limit for PMAC's I^2T integrated current limiting function. If Ix58 is 0, the I^2T limiting function is disabled. If Ix58 is greater than 0, PMAC will compared the time-integrated difference between the squares of commanded current and the Ix57 continuous current limit to Ix58. If the integrated value exceeds Ix58, then PMAC faults the motor just as it would for receiving an amplifier fault signal, setting both the amplifier-fault and the I^2T-fault motor status bits.

The Ix58 limit is typically set by taking the relationship between the instantaneous current limit (Ix69 on PMAC, in units of a 16-bit DAC), the magnetization current (Ix77; typically 0 except for vector control of induction motors) and the continuous current limit (Ix57 on PMAC, in units of a 16-bit DAC) and multiplying by the time permitted at the instantaneous limit. The formula is:

$$Ix58 = \frac{Ix69^2 + Ix77^2 - Ix57^2}{32768^2} * ServoUpdateRate(Hz) * PermittedTime(sec)$$

Refer to the section Making Your Application Safe in the User's Guide for a more detailed explanation of I^2T protection.

Example With the instantaneous current limit Ix69 at 32,767, the magnetization current Ix77 at 0, the continuous current limit Ix57 at 10,589 (1/3 of max), the time permitted with maximum current is at 1 minute, and the servo update rate at the default of 2.25 kHz, Ix58 would be set as:

$$Ix58 = (1.0^2 + 0.0^2 - 0.33^2) * 2250 * 60 = 120000$$

See Also Integrated Current Protection (Making Your Application Safe)

Ix59 Motor x User-Written Servo/Phase Enable

Range 0 .. 3

Units none

Default 0

Remarks Ix59 controls whether the built-in servo and commutation routines, or user-written servo and commutation routines, are used for Motor x.

Ix59	Servo Algorithm	Commutation Algorithm
0	Built-in	Built-in
1	User-written	Built-in
2	Built-in	User-written
3	User-written	User-written

Any user-written servo or commutation (phase) algorithms will have been coded and cross-assembled in a host computer, and downloaded into PMAC's program memory. These algorithms are retained by the battery on battery-backed RAM versions, or saved into flash memory on flash-backed versions.

Ix00 must be 1 in order for the user-written servo to execute. Ix01 must be 1 in order for the user-written commutation to execute. The servo algorithm can be changed immediately between the built-in algorithm and a user-written algorithm by changing Ix59. PMAC only selects the phasing algorithm to be used at power-on reset, so in order to change the commutation algorithm, Ix59 must be changed, this new value stored to non-volatile memory with the **SAVE** command, and the board reset.

It is possible to use the user-written algorithms for purposes other than servo or commutation, making them essentially very fast and efficient PLC programs. This is very useful for fast, position-based outputs. Simply load the code, activate an extra "motor" with Ix00 and/or Ix01, and set Ix59 for this pseudo-motor to use this algorithm.

See Also User-Written Servo Instructions (Closing the Servo Loop)
User-Written Commutation Instructions (Setting Up PMAC Commutation)
I-Variables Ix00, Ix01

Ix60 Motor x Servo Cycle Period Extension

Range 0 .. 8,388,607

Units Servo Interrupt Periods

Default 0

Remarks Ix60 permits an extension of the servo update time for Motor x beyond the servo interrupt period, which is controlled by hardware (E3-E6, E29-E33, E98, and master clock). The servo loop will be closed every (Ix60 + 1) servo interrupts. With the default value of zero, the loop will be closed every servo interrupt. For the standard PID servo algorithm, Ix60 must be set to a value that can be expressed as $(2^n - 1)$, where "n" is a non-negative integer.

Other update times, including trajectory update and phase update are not affected by Ix60. I10 does not need to be changed with Ix60.

The velocity values reported for a motor with the **V** or **<CTRL-V>** command, and the actual velocity registers in regular memory or DPRAM, are affected by Ix60. They are reported in counts per servo loop closure, not counts per servo interrupt.

See Also Servo Update Rate (Closing the Servo Loop)
On-line commands **<CTRL-V>**, **V**
M-Variable Mx66
Jumpers E3-E6, E29-E33, E98; I10.

Ix61 Motor x Current Loop Integral Gain {PMAC2 only}

Range 0.0 .. 1.0 (24-bit resolution)

Units Output = 8 * Ix61 * Sum [i=0 to n] (Icmd[i]-Iact[i])

Default 0

Remarks Ix61 is the integral gain term of the digital current loops, multiplying the difference between the commanded and actual current levels and adding the result into a running integrator that adds into the command output. It is only used if Ix82>0 to activate digital current loop execution.

Ix61 can be used with either Ix62 forward-path proportional gain, or Ix76 back-path proportional gain. If used with Ix62, the value can be quite low, because Ix62 provides the quick response, and Ix61 just needs to correct for biases. If used with Ix76, Ix61 is the only gain that responds directly to command changes, and it must be significantly higher to respond quickly.

Ix61 is typically set using the current loop auto-tuner or interactive tuner in the PMAC Executive Program. Typical values of Ix61 are 0.02.

See Also Setting Up PMAC Commutation
I-variables Ix62, Ix66, Ix76, Ix82

Ix62 Motor x Current Loop Proportional Gain (Forward Path) {PMAC2 only}

Range 0.0 .. 1.0 (24-bit resolution)

Units Output = 4 * Ix62 * (I_{cmd} - I_{act})

Default 0

Remarks Ix62 is the proportional gain term of the digital current loops that is in the "forward path" of the loop, multiplying the difference between the commanded and actual current levels. Either Ix62 or Ix76 (back path proportional gain) must be used to close the current loop. Generally, only one of these proportional gain terms is used, although both can be. Ix62 is only used if Ix82>0 to activate digital current loop execution.

Ix62 can provide more responsiveness to command changes from the position/velocity loop servo, and therefore a higher current loop bandwidth, than Ix76. However, if the command value is very noisy, which can be the case with a low-resolution position sensor, using Ix76 instead can provide better filtering of the noise.

Ix62 is typically set using the current loop auto-tuner or interactive tuner in the PMAC Executive Program. . Typical values of Ix62, when used, are around 0.5.

See Also Setting Up PMAC Commutation
I-variables Ix61, Ix66, Ix76, Ix82

Ix63 Motor x Integration Limit

Range -8,388,608 .. 8,388,607

Units (Ix33/2^{19}) counts * servo-cycles

Default 4,194,304

Remarks Ix63 limits the magnitude of the integrated position error (the output of the integrator), which can be useful for "anti-windup" protection. The default value of Ix63 provides essentially no limitation. (The integral gain Ix33 controls how fast the error is integrated.)

A value of zero here forces a zero output of the integrator, effectively disabling the integration function in the PID filter. This can be useful during periods when you are applying a constant force and are expecting a steady-state position error. (In contrast, setting Ix33 to 0 prevents further inputs to the integrator, but maintains the output.)

The Ix63 integration limit can also be used to create a fault condition for the motor. If Ix63 is set to a negative number, then PMAC will also check as part of its following error safety check whether the magnitude of integrated following error has saturated at the magnitude of Ix63. With Ix63 negative, if the integrator has saturated, PMAC will trip (kill) the motor with a following error fault. Both the normal fatal following error motor status bit and the integrated following error status bit are set when this fault occurs. If Ix63 is 0 or positive, the motor cannot trip on integrated following error fault.

To set Ix63 to a value such that the integrator saturates at the same point that its contribution to the command output causes saturation at the Ix69 level, use the following formula:

$$Ix63 = \pm\left(\frac{Ix69 * 2^{23}}{Ix08 * Ix30}\right)$$

To cause trips, the magnitude of Ix63 must be set to less than this value due to other potential contributions to the output. Remember that the integrator stops increasing when the output saturates at Ix69.

See Also PID Servo Filter (Closing the Servo Loop)
I-variables Ix33, Ix67, Ix69

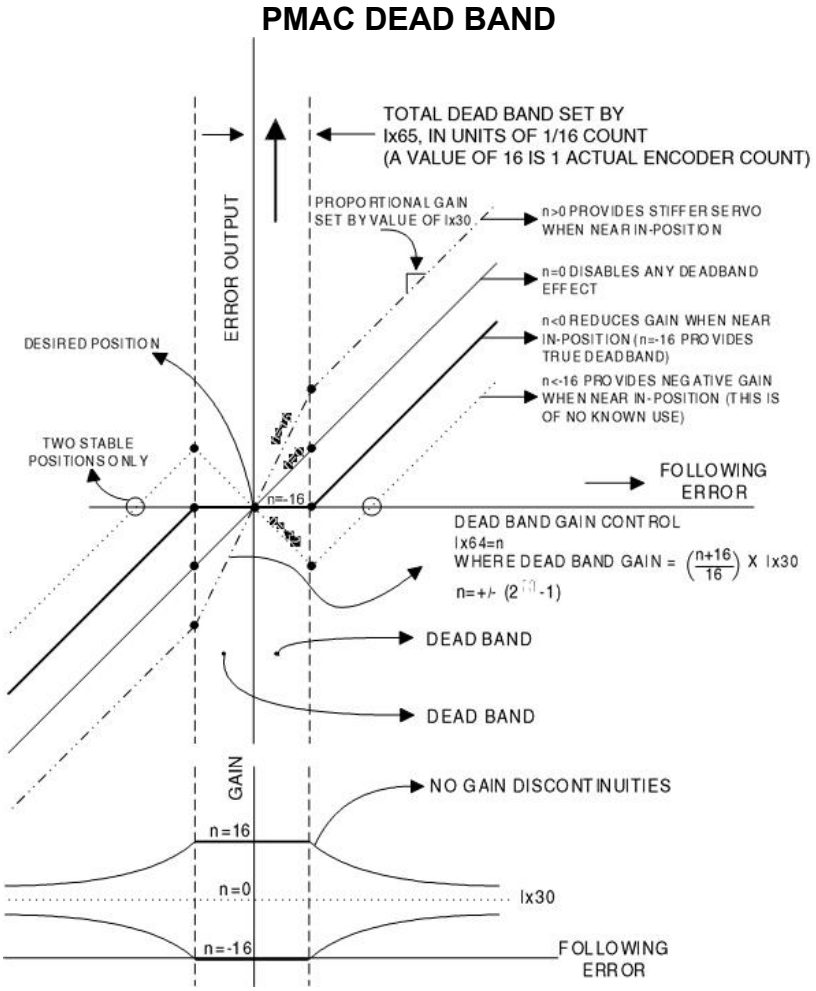
Ix64 Motor x Deadband Gain Factor

Range -32,768 .. 32,767

Units none

Default 0 (no deadband)

Remarks Ix64 is part of the PMAC feature known as deadband compensation, which can be used to create or cancel deadband. It controls the effective gain within the deadband zone (see Ix65). When the following error is less than the value of Ix65, the proportional gain (Ix30) is multiplied by (Ix64+16)/16. At a value of -16, Ix64 provides true deadband. Values between -16 and 0 yield reduced gain within the deadband. Ix64 = 0 disables any deadband effect.



Values of Ix64 greater than 0 yield increased gain within the deadband; a value of 16 provides double gain in the "deadband". A small band of increased gain can be used to reduce errors while holding position, without as much of a threat to make the system unstable. It is also useful in compensating for physical deadband in the system.

Note:

Values of Ix64 less than -16 will cause negative gain inside the deadband, making it impossible for the system to settle inside the band. These settings have no known useful function.

Outside the deadband, gain asymptotically approaches Ix30 as the following error increases.

See Also I-variables Ix30, Ix65
Closing The Servo Loop

Ix65 Motor x Deadband Size

Range 0 .. 32,767

Units 1/16 count

Default 16 (=1 count)

Remarks Ix65 defines the size of the position error band, measured from zero error, within which there will be changed or no control effort, for the PMAC feature known as deadband compensation. Ix64 controls the effective gain relative to Ix30 within the deadband.

> ***Note:***
>
> The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts in the deadband. For example, if modified gain is desired in the range of +/-5 counts of following error, Ix65 should be set to 80.

See Also Deadband (Closing the Servo Loop)
I-variables Ix30, Ix64

Ix66 Motor x PWM Scale Factor {PMAC2 only}

Range 0 .. 32,767

Units PWM_CLK cycles

Default 0

Remarks Ix66 multiplies the output of the digital current loops for Motor x (which are values between -1.0 and 1.0) before they are written to the PWM output registers. As such, it determines the maximum value that can be written to the PWM output register. Ix66 is only used if Ix82>0 to activate digital current loop execution.

The PWM output value for each phase is compared digitally to the PWM up-down counter, which increments or decrements once per PWM_CLK cycle to determine whether the outputs are on or off. The limits of the up-down counter are I900+1 and -I900-2 for channels 1 to 4; I906+1 and -I906-2 for channels 5 to 8.

Generally, Ix66 is set to about 10% above I900 (or I906) for motors commutated by PMAC2. This permits a full-on command of the phase for a substantial fraction of the commutation cycle, providing maximum possible utilization of the power devices at maximum command. If Ix66 is set to a smaller value than I900 or I906, it serves as a voltage limit for the motor (Vmax = I900/Ix66*VDC). Ix69 serves as the current limit.

Ix67 Motor x Linear Position Error Limit

Range 0 .. 8,388,607

Units 1/16 count

Default 4,194,304 (=262,144 counts)

Remarks Ix67 defines the biggest position error that will be allowed into the servo filter. This is intended to keep extreme conditions from upsetting the stability of the filter. However, if it is set too low, it can limit the response of the system to legitimate commands (this situation can particularly be noticed on very fine resolution systems).

This parameter is not to be confused with Ix11 or Ix12, the following error limits. Those parameters take action outside the servo loop based on the real (before limiting) following error.

Note:

The units of this parameter are 1/16 of a count, so the value should be 16 times the number of counts in the limit.

See Also I-variables Ix11, Ix12, Ix68

Ix68 Motor x Friction Feedforward

Range -32,768 .. 32,767

Units DAC bits

Default 0

Remarks Ix68 adds a bias term to the servo loop output of Motor x that is proportional to the *sign* of the commanded velocity. That is, if the commanded velocity is positive, Ix68 is added to the output. If the commanded velocity is negative, Ix68 is subtracted from the output. If the commanded velocity is zero, no value is added to or subtracted from the output.

This parameter is intended primarily to help overcome errors due to mechanical friction. It can be thought of as a "friction feedforward" term. Because it is a feedforward term that does not utilize any feedback information, it has no direct effect on system stability. It can be used to correct the error resulting from friction, especially on turnaround, without the time constant and potential stability problems of integral gain.

If PMAC is commutating this motor, this correction is applied before the commutation algorithm, and so will affect the magnitude of both analog outputs.

Note:

This direction-sensitive bias term is independent of the constant bias introduced by Ix29 and/or Ix79.

Example Starting with a motor at rest, if Ix68 = 1600, then as soon as a commanded move in the positive direction is started, a value of +1600 (~0.5V) is added to the servo loop output. As soon as the commanded velocity goes negative, a value of -1600 is added to the output. When the commanded velocity becomes zero again, no bias is added to the servo output as a result of this term.

See Also Closing the Servo Loop
I-Variables Ix01, Ix29, Ix32, Ix35, Ix79

Ix69 Motor x Output Command Limit

Range 0 .. 32,767 (0 to 10V or equivalent)

Units 16-bit DAC bits

Default 20,480 (6.25V or equivalent)

Remarks Ix69 defines the magnitude of the largest output that can be sent from PMAC's PID position/velocity servo loop. If the servo loop computes a larger value, PMAC clips it to this value. When the PID output has saturated at the Ix69 limit, the integrated error value will not increase, providing anti-windup protection.

For the Extended Servo Algorithm (ESA) that with the Option 6 firmware version, Ix69 is used to multiply a normalized command (-1.0 <= Normalized Command < +1.0) before outputting it or using it for commutation. As such, it acts as both a scale factor and an output command limit for the ESA.

Ix69 is always in units of a 16-bit DAC, even if the actual output device is of a different resolution, or the command value is used for PMAC's own internal current loop commands.

If you are using differential analog outputs (DAC+ and DAC-), the voltage between the two outputs is twice the voltage between an output and AGND, so the Ix69 value should be set to half of what it would be for a single-ended analog output.

This parameter provides a torque (current) limit in systems with current-loop amplifiers, or when using PMAC's internal commutation; it provides a velocity limit with velocity-mode amplifiers. Note that if this limit "kicks in" for any amount of time, the following error will start increasing.

Use when Commutating: When PMAC is commutating Motor x (Ix01 = 1) but not closing the current loops (Ix82 = 0), Ix69 corresponds to *peak* values of the sinusoidal phase currents. Motor and amplifier current limits are usually given as RMS values. Peak phase values are √2, or 1.414, times greater than RMS values. For instance, if an amplifier has a 10 amp (RMS) instantaneous current limit, the instantaneous limit for the peak of the phase currents is 14.14 amps.

Use with Magnetization Current: When commutating, Ix69 is technically the limit of only the quadrature, or torque-producing, current. Ix77 sets the magnitude of the direct, or magnetization current, and the total current limit is the vector sum of these two variables. If the Ix77 magnetization current for the motor is set to a value other than 0, Ix69 should be set such that:

$$\sqrt{Ix69^2 + Ix77^2} \leq I_{max} \leq 32{,}767$$

Use in Direct-PWM Mode: When commutating (Ix01 = 1) and closing the current loop (Ix82 > 0) of a 3-phase motor (Ix72 = 85 or 171), it is important to understand the relationship between the quadrature current limited by Ix69 and the phase currents measured by the A/D converters. This difference is due to the nature of the conversion between direct and quadrature current components, which are 90° apart, and the phase currents, which are 120° apart. This difference introduces a factor of cos(30°) into the calculations.

For a given level of DC quadrature current with zero direct (magnetization) current, the peak value of AC sinusoidal current measured in the phases will be 1/cos(30°), or 1.15 times, greater. When quadrature current is commanded at its limit of Ix69, the peak phase currents can be 15% higher that this value. For instance, with Ix69 at 10,000, and Ix77 at 0, the A/D converters can provide readings (normalized to 16-bit resolution) up to 11,547.

Use with Magnetization Current: With non-zero direct current, the peak value of AC sinusoidal current measured in the phases will be 1.15 times greater than the vector sum of the direct and quadrature currents. Therefore, in order not to saturate the current in the phases, Ix69 should be set such that:

$$\sqrt{Ix69^2 + Ix77^2} \leq I_{max}\cos(30^o) \leq 32{,}767 * 0.866 \leq 28{,}377$$

Example

1. Motor 1 is driving a velocity-mode amplifier with differential analog inputs that are limited to +/-10V between the inputs. This means that the PMAC outputs should each be limited to +/-5V with respect to the AGND reference. I169 should therefore be limited to 32,768/2 = 16,384.
2. Motor 3 is driving a DC brush motor amplifier in current (torque) mode with an analog output. The amplifier has a gain of 2 amps/volt and an instantaneous current limit of 20 amps. The motor has an instantaneous current limit of 15 amps.

- PMAC's maximum output of 32,768, or 10 volts, corresponds to 20 amps.
- The motor has the lower instantaneous current rating, so we use its limit of 15 amps.
- I369 is set to 32,768 * 15/20 = 24,576.

3. Motor 5 is driving a self-commutating brushless-motor amplifier in current (torque) mode with a single analog output. The amplifier has a gain of 5 amps(RMS)/volt and an instantaneous current limit of 50 amps (RMS). The motor has an instantaneous current limit of 60 amps (RMS).
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 50 amps (RMS).
   - The amplifier has the lower instantaneous current rating, so we use its limit of 50 amps (RMS).
   - I569 is set to 32,768 * 50/50 = 32,767 (note that the maximum value is 32,767).
4. Motor 7 is driving a "sine-wave" amplifier for a brushless servo motor with two analog outputs. The Ix77 magnetization current limit is set to 0. The amplifier has a gain on each phase of 4 amps/volt. The amplifier has an instantaneous current limit of 25 amps (RMS). The motor has an instantaneous current limit of 30 amps (RMS).
   - PMAC's maximum output of 32,768, or 10 volts, corresponds to 40 amps peak in the phase.
   - The amplifier has the lower instantaneous current rating, so we use its limit of 25 amps (RMS).
   - 25 amps (RMS) corresponds to peak phase currents of 25*1.414 = 35.35 amps.
   - I769 is set to 32,768 * 35.35/40 = 28,958.
5. Motor 8 is driving a direct-PWM "power-block" amplifier and an AC induction motor. The Ix77 magnetization current parameter is set to 3000. The A/D converters in the amplifier are scaled so that a maximum reading corresponds to 100 amps of current in the phase. The amplifier has an instantaneous current limit of 60 amps (RMS), and the motor has an instantaneous current limit of 75 amps (RMS).
   - PMAC's maximum ADC phase reading of 32,768 corresponds to 100 amps in the phase.
   - The amplifier has the lower instantaneous current rating, so we use its limit of 60 amps (RMS).
   - 60 amps (RMS) correspond to peak phase currents of 60*1.414 = 84.84 amps.
   - 84.84 amps correspond to an ADC reading of 32,768 * 84.84/100 = 27,800.
   - The vector sum of Ix69 and Ix77 should equal 27,800 * 0.866 = 24,075.
   - I869 should be set to sqrt($24{,}075^2$-$3{,}000^2$) = 23,887.

Commutation I-Variables

Ix70 Motor x Number of Commutation Cycles (N)

Range 0 .. 255

Units Commutation cycles

Default 1

Remarks For a PMAC-commutated motor (Ix01=1), this parameter is used in combination with Ix71 to define the size of the commutation cycle, in encoder counts, as Ix71/Ix70. Usually, this is set to one, and Ix71 represents the number of counts in a single commutation cycle. Ix70 only needs to be set greater than one if the number of counts in a single cycle is not an integer.

A commutation cycle, or electrical cycle, consists of two poles (one pole pair) of a multiphase motor.

Example A 6-pole brushless motor has three commutation cycles per mechanical revolution. If a feedback device is used with 4096 counts per mechanical revolution (a number not divisible by three), Ix70 should be set to 3, and Ix71 to 4096.

See Also I-variables Ix01, Ix71-Ix83
Setting Up PMAC Commutation

Ix71 Motor x Encoder Counts per N Commutation Cycles

Range 0 .. 8,388,607

Units Counts

Default 1000

Remarks For a PMAC-commutated motor, Ix71 defines the size of a commutation cycle in conjunction with Ix70 (counts/cycle = Ix71/Ix70). The meaning of a "count" used in this parameter is defined by the encoder-decode variable for the commutation feedback device (Encoder I-Variable 0; I900, I905, etc. on a PMAC(1); I9n0 on a PMAC2). If a "times-4" decode is used, a *count* is one-fourth of an encoder *line*.

If a highly interpolated encoder is used (e.g. from an ACC-51P or ACC-8D Opt 8) for servo loop closure, the digital hardware quadrature counter is usually still used for commutation, with a resolution of 4 counts per encoder line.

If the commutation feedback comes from a MACRO-node position feedback register, the position value is usually in units of 1/32 of a count, so Ix71 should be 32 times larger than it would be for reading a hardware encoder counter directly.

A commutation cycle, or electrical cycle, consists of two poles (one pole pair) of a multiphase motor.

Example A four-pole brushless motor with a 1000-line-per-revolution encoder and "times-4" decode read directly on the PMAC has 2 commutation cycles per revolution and 4000 counts per revolution. Therefore, either Ix70=2 and Ix71=4000 could be used, or Ix70=1 and Ix71=2000.

For the same motor and encoder read through a MACRO Station, the units of the position register read for commutation would be 1/32-count, so there would appear to be 4000*32, or 128,000 counts per revolution. Therefore, either Ix70=2 and Ix71=128000 could be use, or Ix70=1 and Ix71=64000.

See Also I-variables Ix01, Ix70, Ix72-Ix83
Setting Up PMAC Commutation

Ix72 Motor x Commutation Phase Angle

Range 0 .. 255

Units 360/256 elec. deg. (1/256 commutation cycle)

Default 85 (=120^{o} e)

Remarks For a PMAC-commutated motor (Ix01 = 1), Ix72 set the angular distance between the phases of a multiphase motor. The usual values to be used are:

3-phase: 85 or 171 (+/- 120^{o}e)

2- or 4-phase: 64 or 192 (+/- 90^{o}e)

For a given number of phases, determining which of the two possible settings is to be used depends on whether the PMAC is also closing the current loop for the motor.

1. PMAC performing commutation, but not current loop
When PMAC *is not* performing digital current loop closure for Motor x (Ix82 = 0), the output direction sense determined by this parameter and the motor and amplifier phase wiring must match the feedback direction sense as determined by the encoder-decode variable 0 (I900, I905, etc. on a PMAC(1); I9n0 on a PMAC2) and the encoder wiring. If the direction senses do not match proper commutation and servo control will be impossible; the motor will lock into a given position.

For these systems, changing between the two values for a given number of phases has the same effect as exchanging motor leads, which changes the motor's direction of rotation for a given sign of a PMAC2 torque command.

Refer to the section *Setting Up PMAC Commutation* for tests to determine the proper Ix72 setting. For systems without PMAC2 digital current loop closure, once this commutation/feedback polarity has been properly matched, the servo/feedback polarity will automatically be properly matched.

2. PMAC performing commutation and current loop
When PMAC *is* performing digital current loop closure for Motor x (Ix82 > 0; PMAC2 only), the output direction sense determined by this parameter must match the polarity of the phase current sensors and the analog-to-digital conversion (ADC) circuitry that brings this data into PMAC2. It is independent of motor or amplifier phase wiring, encoder wiring, and PMAC2 encoder-decode direction sense.

CAUTION:

Do not attempt to close the digital current loops on PMAC2 (O commands or closing the position loop) until you are sure of the proper sense of the Ix72 setting. An Ix72 setting of the wrong sense will cause positive feedback in the current loop, leading to saturation of the PMAC outputs and possible damage to the motor and or amplifier.

For these systems with a PMAC2 digital current loop, if the phase-current ADC registers report a positive value for current flowing *into* the phase (i.e. the PWM voltage command value and the current feedback value have the same sign), Ix72 must be set to a value greater than 128 (usually 171 for a 3-phase motor, or 192 for a 2- or 4-phase motor). If the phase-current ADC registers report a positive value for current flowing out *of* the phase (i.e. the PWM voltage command value and the current feedback value have opposite signs), Ix72 must be set to a value less than 128 (usually 85 for a 3-phase motor, or 64 for a 2- or 4-phase motor).

For systems with PMAC2 digital current loop closure, the commutation/feedback polarity match is independent of the servo/feedback polarity. Once Ix72 has been set for proper commutation/feedback polarity, the proper position-loop servo/feedback polarity must still be established.

See Also I-variables Ix70, Ix71
Encoder I-Variable 0
Setting Up PMAC Commutation
Getting Started Section, Setting Up A PMAC-Commutated Motor

Ix73 Motor x Phase Finding Output Value

Range -32,768 .. 32,767

Units Bits of 16-bit DAC

Default 0

Remarks

WARNING:

An unreliable phasing search method can lead to a runaway condition. Test your phasing search method carefully to make sure it works properly under all conceivable conditions. Make sure your Ix11 fatal following error limit is active and as tight as possible so the motor will be killed quickly in the event of a serious phasing search error.

Ix73 defines the magnitude of the open-loop output to be used if a power-on phasing search is done for a PMAC-commutated motor (Ix01=1). A phasing search is required for a synchronous motor (Ix78=0) such as a permanent-magnet brushless motor with no absolute position sensor (Ix81=0). The phasing search is done automatically as part of the power-on phasing search if Ix80 is 1 or 3; if Ix80 is 0 or 2, the on-line **$** command must be used must be used to initiate the phasing search.

If Ix80 is 0 or 1, the two-guess phasing search is used, and Ix73 controls the "vector" magnitude of the open-loop output that is distributed among the phases according to the guessed phasing angle.

If Ix80 is 2 or 3, the stepper-motor phasing search is used, and Ix73 controls the magnitude of current forced into individual phase(s) to lock the motor to a position like a stepper motor. In this method, if the PMAC2 is not performing current loop closure for the motor (Ix82 = 0) and Ix72 > 128, then Ix73 should be set to a negative number of the desired magnitude. In all other cases it should be set to a positive number. If the sign of Ix73 is wrong for your setup, the motor will run away when the loop is closed.

Values of 2000 to 6000 are typically used for Ix73 in either method.

See Also Power-Up Phasing Search (Setting Up PMAC Commutation)
I-Variables Ix01, Ix74, Ix78, Ix80, Ix81

Ix74 Motor x Phase Finding Time

Range 0 .. 255

Units Servo Interrupt Cycles (for Ix80 = 0 or 1)
or
Servo Interrupt Cycles * 256 (for Ix80 = 2 or 3)

Default 0

Remarks

WARNING:

An unreliable phasing search method can lead to a runaway condition. Test your phasing search method carefully to make sure it works properly under all conceivable conditions. Make sure your Ix11 fatal following error limit is active and as tight as possible so the motor will be killed quickly in the event of a serious phasing search error.

Ix74 defines the time that an open-loop output is to be used if a power-on phasing search is done for a PMAC-commutated motor (Ix01=1). A phasing search is required for a synchronous motor (Ix78=0) such as a permanent-magnet brushless motor with no absolute position sensor (Ix81=0). The phasing search is done automatically as part of the power-on phasing search if Ix80 is 1 or 3; if Ix80 is 0 or 2, the on-line **$** command must be used must be used to initiate the phasing search.

If Ix80 is 0 or 1, the "two-guess" phasing search is used; Ix74 has units of servo cycles and controls the time for the open-loop command at each "guess" of the phase angle. Typical values are 3 to 10 servo cycles; 5 is a good starting point.

If Ix80 is 2 or 3, the "stepper-motor" phasing search is used; Ix74 has units of (servo cycles*256) and controls the time current is forced into each phase and PMAC waits for the motor to settle into the "step" position. With the default servo cycle rate of 2.25 kHz, each unit of Ix74 represents about 0.1 seconds in this mode; typical values are 10 to 20.

See Also Power-Up Phasing Search (Setting Up PMAC Commutation)
I-Variables Ix01, Ix73, Ix78, Ix80, Ix81

Ix75 Motor x Power-On Phase Position Offset

Range -8,388,608 .. 8,388,607

Units Encoder counts * Ix70

Default 0

Remarks Ix75 tells PMAC the distance between the zero position of an absolute sensor used for power-on phase position (specified by Ix81) and the zero position of PMAC's commutation cycle. It is used to reference the phasing algorithm for a PMAC-commutated motor with an absolute sensor (Ix81 > 0). If Ix80 is 1, this is done automatically during the power-up/reset cycle. It will also be done in response to a **$** command to the motor.

The **SETPHASE** command also uses Ix75, copying the Ix75 value directly into the phase position register. This mode is typically used to correct the phasing at a known position (usually at the index pulse of the encoder) after a rough phasing (hall-sensor read or phasing search) gives you enough torque for basic motion to the known position.

The proper value for this parameter can be found with a simple procedure that should be done with an unloaded motor, after satisfactory operation has been achieved using a power-on phasing search.

For use with the Ix81 absolute read, define an M-variable to the absolute sensor (TWR form for a resolver, Y form for an absolute encoder). Next, drive the motor to the zero position in the commutation cycle, either by issuing a $ command with the motor set up for the "stepper motor" phasing search (Ix80 = 1 or 3), or by manually setting the phase offsets for the motor.

In the manual technique, give the motor an **O0** command. Put a bias on the A phase (higher-numbered DAC of a PMAC1 pair) by setting Ix29; use a positive bias if Ix72=171 or 192 (2000 is usually a good value); use a negative bias if Ix72=85 or 64. Also put a bias in the opposite direction of the same magnitude on the B phase by setting Ix79. The motor should lock in on a position like a stepper motor.

Now remove the A-phase bias by setting Ix29 back to zero, or at least to the value you have found to force zero current in the phase, and the motor should lock in on another position. This position is the zero position of the phasing cycle.

In either technique for forcing the motor to its zero commutation position, after you are sure the motor has settled, read the position of the absolute sensor by querying its M-variable value.

Take the negative of this value, multiply it by Ix70, and put the resulting value in Ix75. Now, with Ix79 returned to zero or the proper bias, and Ix81 pointing to the absolute sensor, give the motor a **$** command. The motor should be properly phased. Remember to save these variable values before doing a full reset on the card.

For use with the **SETPHASE** command, define an M-variable to the the phase position register. The suggested M-variable is Mx71 (e.g. **M171->X:$0041, 0,24,S**). Execute the above sequence with Ix29 and Ix79 to force the motor to the zero-point in its phase cycle. Set the M-variable to zero (e.g. **M171=0**). Now move the motor to the known position in its cycle (usually with a homing search move), let it settle, and read the M-variable value. This value will be put in Ix75.

Example

On a brushless motor #1 commutated from PMAC with Ix70 =1 and Ix72 = 171, using an R/D converter at location 0 of a board at multiplexer address 0, the following on-line commands can be used to set Ix75:

```
M171->TWR:0,0          ;Resolver position
#1O0                   ;Open-loop zero command
I129=2000              ;Pos bias on first phase output (Neg if Ix72=85 or 64)
I179=-2000             ;Neg bias on second phase output (Pos if Ix72=85 or 64)
I129=0                 ;Remove bias from first phase output
M171                   ;Query sensor position
223                    ;PMAC responds
I175=-223              ;Set phasing position offset (223*-1*Ix70)
I179=0                 ;Remove bias from second phase
I181=$000100           ;Set power-on position address
I173=0                 ;Make sure no phasing search move is done
I174=0                 ;Make sure no phasing search move is done
SAVE                   ;Store I-variables in non-volatile memory
$                      ;Try phasing from absolute position sensor
```

See Also

Phasing Referenced to Absolute Sensor (Setting Up PMAC Commutation)
I-Variables I8x, I9x, Ix03, Ix10, Ix81, Ix83
ACC-8D Option 7 (R/D Converter) Manual

Ix76 Motor x Velocity Phase Advance Gain {PMAC(1) Only}

Range 0 .. 8,388,607

Units (Ix09*32) / (Ix70*2^{23}) counts / (counts per servo update)

Default 0

Remarks Ix76 advances the phasing angles on a motor commutated by PMAC(1) by an amount proportional to the measured velocity of the motor. This compensates for the lag in the electrical circuits of the phases, and for calculation delays. It should be set to zero for induction motors.

This parameter is best set experimentally by running the motor at high speeds, and finding the setting that minimizes the current draw of the motor.

See Also Setting Up PMAC Commutation
I-variables Ix70-Ix75

Ix76 Motor x Current-Loop Proportional Gain (Back Path) {PMAC2 only}

Range 0.0 .. 1.0 (24-bit resolution)

Units PWMout = -4 * Ix76 * (Iact)

Default 0.0

Remarks Ix76 is the proportional gain term of the digital current loop that is in the back path of the loop, multiplying the actual current level, and subtracting the result from the command output. Either Ix76 or Ix62 (forward path proportional gain) must be used to close the current loop. Generally, only one of these proportional gain terms is used, although both can be.

If Ix76 is used as the only proportional gain term, only the Ix61 integral gain term reacts directly to command changes. The act of integration acts as a low-pass filter on the command, which eliminates a lot of noise, but lowers the responsiveness to real changes. Generally Ix76 is only used when the command value from the position/velocity loop servo have high noise levels (usually due to low position resolution), and the actual current measurements have low noise levels.

Typically, Ix76 is set using the current loop auto-tuner or interactive tuner in the PMAC Executive Program. Typical values of Ix76, when used, are around 0.5.

Ix76 is only used if Ix82>0 to activate digital current loop execution.

See Also Setting Up PMAC Commutation
I-variables Ix61, Ix62, Ix66, Ix82

Ix77 Motor x Induction Motor Magnetization Current

Range -32,768 .. 32,767

Units DAC bits

Default 0

Remarks Ix77 is used in induction motors to provide a stator current component parallel to the estimated rotor magnetic field (the "direct" current – the control loop determines the magnitude of the "quadrature" current perpendicular to this component). This should be set to zero for non-induction motors. The proper value for an induction motor is system dependent, but 2500 is a good starting value for most motors. Refer to the Setting Up PMAC Commutation section of the manual for instructions in optimizing the setting of this parameter.

See Also Setting Induction Motor Parameters (Setting Up PMAC Commutation)
I-variables Ix01, Ix70-Ix72, Ix78

Ix78 Motor x Induction Motor Slip Gain

Range 0 .. 8,388,607 {PMAC(1)}
0.0.. 1.0 (24-bit resolution) {PMAC2}

Units 2^{38} (electrical cycles/update)/DAC bit {PMAC(1)}
Unitless (ratio of times) {PMAC2}

Default 0

Remarks Ix78 controls the relationship between the torque command and the slip frequency of magnetic field on the rotor of an AC asynchronous (induction) motor. While it is usually set experimentally, it can be calculated as the ratio between the phase update period and the rotor (not stator) L/R electrical time constant.

Ix78 is only active if Ix01 is set to 1 to specify PMAC2 commutation of Motor x. It should be set to 0 for AC synchronous motors such as permanent-magnet brushless motors and switched (variable) reluctance motors.

Ix78 operates slightly differently on PMAC(1) and PMAC2 boards.

PMAC(1) computes the slip frequency each phase update by multiplying the torque command from the position/velocity-loop servo (or O-command magnitude) by Ix78. The optimum value of Ix78 is dependent of the value of the Ix77 magnetization current, so if Ix77 is changed (e.g. for field weakening), Ix78 should be changed in opposite proportion so that the product of Ix77 and Ix78 stays constant.

PMAC2 computes the slip frequency each phase update by multiplying the torque command from the position/velocity-loop servo (or O-command magnitude) by Ix78 and then dividing by the magnetization current value controlled by Ix77. This makes the optimum value of Ix78 independent of the value of Ix77, so changing the value of Ix77 for field control does not require changes in Ix78.

Ix78 is typically set on a PMAC2 through use of the P2SETUP expert system program running on a PC. P2SETUP excites the motor and analyzes its response to derive an optimum Ix78 value.

Ix78 can also be set experimentally by giving the motor an O-command and watching the velocity response, probably with the data gathering feature. As the velocity saturates because the back EMF reaches the supply voltage, the velocity should fall back about 5% to reach a steady-state value. If it falls back more than this, the slip time constant is too high; if it falls back less than this, or not at all, the slip time constant is too low.

On a PMAC(1), 1200 is a typical value of Ix78 for an standard induction motor at a phase update rate of about 9 kHz.

On a PMAC2, 0.00015 is a typical value of Ix78 for an standard induction motor at a phase update rate of about 9 kHz.

If Ix78 is greater than zero, no power-on phasing search will be done (because the rotor field is not fixed to the rotor).

See Also Setting Induction Motor Parameters (Setting Up PMAC Commutation)
I-Variables Ix01, Ix70-Ix72, Ix77

Ix79 Motor x Second Phase Offset

Range -32,768 .. 32,767

Units 16-bit DAC/ADC bit equivalent

Default 0

Remarks Ix79 serves as an output or feedback offset for Motor x; its exact use depends on the mode of operation as described below:

Mode 1: When PMAC is not commutating Motor x (Ix01 = 0) and the output is bipolar (Ix02 bit 16 = 1, the default), Ix79 is not used. Ix29 is the offset for this mode.

Mode 2: When PMAC is not commutating Motor x (Ix01 bit 0 = 0) but is in sign-and-magnitude output mode (Ix02 bit 16 = 1 – PMAC(1) only), Ix79 is the offset of the command output value after the absolute value is taken (Ix29 is the offset before the absolute value is taken). Ix79 is typically used in this mode to compensate for analog offsets in interface circuitry, either in DACs or in voltage-to-frequency converters.

Mode 3: When PMAC is commutating Motor x (Ix01 = 1) but not closing the current loop (Ix82 = 0), Ix79 serves as the offset for the second of two phase command output values (Phase B), for the address specified by Ix02 plus 1; Ix29 serves the same purpose for the first phase. Ix79 is added to the output command value before it is written to the command output register.

When commutating from a PMAC(1), Phase A is output on the *higher*-numbered of the two DACs (e.g. DAC2), Phase B on the *lower*-numbered (e.g. DAC1). When commutating from a PMAC2, Phase A is output on the A-channel DAC (e.g. DAC1A), Phase B on the B-channel DAC (e.g. DAC1B).

As an output command offset, Ix79 is always in units of a 16-bit register, even if the actual output device is of a different resolution. For example, if a value of 60 had to be written into an 18-bit DAC to create a true zero command, this would be equivalent to a value of 60/4=15 in a 16-bit DAC, so Ix79 would be set to 15 to cancel the offset.

Mode 4: When PMAC is commutating (Ix01 = 1) and closing the current loop for Motor x (Ix82 > 0), Ix79 serves as an offset that is added to the phase current reading from the ADC for the second phase (Phase B), at the address specified by Ix82. Ix29 performs the same function for the first phase. The sum of the ADC reading and Ix79 is used in the digital current loop algorithms.

As an input feedback offset, Ix79 is always in units of a 16-bit ADC, even if the actual ADC is of a different resolution. For example, if a 12-bit ADC reported a value of -5 when no current was flowing in the phase, this would be equivalent to a value of -5*16=-80 in a 16-bit ADC, so Ix79 would be set to 80 to compensate for this offset.

Ix80 Motor x Power-Up Mode

Range 0 .. 7

Units None

Default 0

Remarks Ix80 controls the power-up mode, including the phasing search method (if used), for Motor x. It consists of 3 independent control bits, each determining one aspect of the state of the motor at power-up or full board reset:

- Bit 0 controls whether the motor is enabled at power-up/reset or not. If bit 0 is set to 0, the motor is left in the "killed" (disabled) state at power-up/reset, and a command must be issued to the motor to enable it. If bit 0 is set to 1, the motor is automatically enabled at power-up/reset, and if a phasing search move is required to establish the commutation position reference, this is automatically done.

- Bit 1 controls what type of phasing search move is performed, if one is required (Ix74 > 0), either during power-up/reset, or on a subsequent **$** motor reset command. If bit 1 is 0 and a phasing search move is required, PMAC will use the two-guess phasing search method. If bit 1 is 1 and a phasing search move is required, PMAC will use the "stepper-motor" phasing search method. The state of bit 1 does not matter unless a phasing search move is to be done.
- Bit 2 controls whether an absolute position read for the motor is done at power-up/reset or not, if one is required (Ix10 > 0). If bit 2 is set to 0 and an absolute position read is specified, this read operation will be performed automatically at the board power-up/reset. If bit 2 is set to 1 and an absolute position read is specified, this read operation will not be done automatically at power-up/reset, and the **$*** command must be issued to perform the absolute position read. The state of bit 2 does not matter unless an absolute position read is to be done.

The possible values of Ix80 and the function of each are described in the following table:

Ix80	Absolute Position Read at Power-up/Reset?	Phasing Search Method	Power-up/Reset Enable State
0	Yes	Two-Guess	Disabled
1	Yes	Two-Guess	Enabled
2	Yes	Stepper-Motor	Disabled
3	Yes	Stepper-Motor	Enabled
4	No	Two-Guess	Disabled
5	No	Two-Guess	Enabled
6	No	Stepper-Motor	Disabled
7	No	Stepper-Motor	Enabled

Power-up/reset enable state: If the motor is not automatically enabled at power-up/reset, a command must be used subsequently to enable the motor. If PMAC is commutating the motor (Ix01 = 1) and it is a synchronous motor (Ix78 = 0), a phase reference must be established with the **$** or **$$** command as part of the enabling process. The motor cannot be enabled before a successful phase reference is established, because the motor "phase reference error" status bit that is automatically set on power-up/reset will not have been cleared.

If the motor is either not commutated by PMAC (Ix01 =0) or it is not a synchronous motor (Ix78 > 0), a simple enabling command can be used. The **J/** command enables a single motor; the **A** command enables all of the motors in a coordinate system; the **<CTRL-A>** command enables all of the motors on PMAC.

The phase reference, whether executed at power-up/reset or on the **$** command, can be done either by reading an absolute position sensor (Ix81 > 0) or by a phasing search move (Ix74 > 0) if only an incremental sensor is used.

WARNING:

An unreliable phasing search method can lead to a runaway condition. Test your phasing search method carefully to make sure it works properly under all conceivable conditions. Make sure your Ix11 fatal following error limit is active and as tight as possible so the motor will be killed quickly in the event of a serious phasing search error.

Phasing search move method: The two-guess phasing search is very quick and requires little movement, but can be adversely affected if there are significant external loads such as friction and gravity. The stepper-motor phasing search takes more time and causes more movement, but it is more reliable in the presence of significant external loads.

Absolute motor position read: If Ix10 is set to 0, the position reference for a motor comes from a homing search move. If Ix10 is greater than 0, the position reference comes from reading an absolute position sensor at the address and with the format specified by Ix10. In this case, Ix80 bit 2 specifies whether or not this read is done automatically at power-up/reset.

If the absolute position read is not done automatically at power-up/reset, the motor position will be set to 0 at this time. This does not prevent full operation of the motor. The **$*** command must be used later to read the sensor and establish absolute position. Even if the absolute position is read automatically at power-up/reset, it may be read again later with the **$*** command.

See Also Power-Up Phasing Search (Setting Up PMAC Commutation)
On-line commands **$**, **$$**, **$***, **$$$**
I-Variables Ix01, Ix73, Ix74, Ix78, Ix81

Ix81 Motor x Power-Up Phase Position Address

Range $000000 .. $FFFFFF

Units Extended PMAC Addresses

Default 0

Remarks Ix81 tells PMAC what address to read for absolute power-on phase-position information for Motor x, and how to read it, if such information is present. This can be a different address from that of the ongoing phase position information, which is specified by Ix83. Ix81 is set to zero if no special power-on phase position reading is desired, as is the case for an incremental encoder.

If Ix81 is set to zero, a power-on phasing search routine is required for synchronous fixed-field brushless motors (permanent magnet, and switched reluctance); those that have a slip gain (Ix78) of zero. PMAC's automatic phasing search routines based on Ix73 and Ix74 can be used, or a custom power-on PLC routine can be written.

Note:

Ix81 is used for PMAC's commutation algorithms alone, to locate position within one electrical cycle of the motor. It is not used for any servo loop position information, even for power-up. Ix10 is used for that purpose.

Ix81 consists of two parts. The low 16 bits contain the address of the register containing the power-on position information, either a PMAC memory-I/O address, or an address on the multiplexer ("thumbwheel") port. The high 8 bits specify how to read the information at this address.

Note:

It is easier to specify this parameter in hexadecimal form ($ prefix). If I9 is set to 2 or 3, the value of this variable will be reported back to the host in hexadecimal form.

The possible value ranges of Ix81 and the position sources they specify are summarized in the following table:

Ix81 Value Range	Absolute Position Source	Ix81 Address Type
$00xxxx - $07xxxx	ACC-8D Opt 7 R/D Converter	Multiplexer Port
$08xxxx - $18xxxx	Parallel Data Y-Register	PMAC Memory-I/O
$48xxxx - $58xxxx	Parallel Data X-Register	PMAC Memory-I/O
$73xxxx	MACRO Station R/D Converter	MACRO Node Number
$74xxxx	MACRO Station Parallel Read	MACRO Node Number
$80xxxx - $FFxxxx	Hall Sensor Read	PMAC Memory-I/O

The following section provides detail for each type of position feedback.

R/D Converter: If Ix81 contains a value from $0000xx to $0700xx, Motor x will expect its absolute power-on phase position from an ACC-8D Opt. 7 R/D converter board. The low 8 bits(last 2 hex digits) of Ix81 should contain the address of the board on the multiplexer port, as set by the DIP switches on the board.

The second hex digit of Ix81, which can take a value from 0 to 7 in this mode, specifies the number of the individual R/D converter at that multiplexer port address. This is a function of the DIP switch setting on the board and the location of the converter on the board, as specified in the following table:

Ix81 Value	ACC-8D Opt. 7 SW1-1 Setting	# of R/D Converter on ACC-8D Opt. 7
$0000xx	CLOSED (0)	1
$0100xx	CLOSED (0)	2
$0200xx	CLOSED (0)	3
$0300xx	CLOSED (0)	4
$0400xx	OPEN (1)	1
$0500xx	OPEN (1)	2
$0600xx	OPEN (1)	3
$0700xx	OPEN (1)	4

The following table shows the value of Ix81 for the multiplexer port addresses for the ACC-8D Opt. 7 that can be used:

Board Mux. Addr.	Ix81	Board Mux. Addr.	Ix81	Board Mux. Addr.	Ix81	Board Mux. Addr.	Ix81
0	$0n0000*	64	$0n0040	128	$0n0080	192	$0n00C0
8	$0n0008	72	$0n0048	136	$0n0088	200	$0n00C8
16	$0n0010	80	$0n0050	144	$0n0090	208	$0n00D0
24	$0n0018	88	$0n0058	152	$0n0098	216	$0n00D8
32	$0n0020	96	$0n0060	160	$0n00A0	224	$0n00E0
40	$0n0028	104	$0n0068	168	$0n00A8	232	$0n00E8
48	$0n0030	112	$0n0070	176	$0n00B0	240	$0n00F0
56	$0n0038	120	$0n0078	184	$0n00B8	248	$0n00F8
'n' is a digit from 0 to 7 specifying the converter number at that address * If 'n' is 0 and the multiplexer address is 0, the 4th hex digit should be set to 1, making Ix81=$000100; otherwise with Ix10=0, no absolute position would be read.							

Parallel Data Read: If Ix81 contains a value from $08xxxx to $18xxxx, or from $48xxxx to $58xxxx, Motor x will do a parallel data read of the PMAC memory or I/O register at the address specified by the low 16 bits of Ix81.

In this mode, bits 16 to 21 of Ix81 specify the number of bits to be read, starting with bit 0 at the specified address. In this mode, they can take a value from $08 to $18 (8 to 24).

In this mode, bit 22 of Ix81 specifies whether a Y-register is to be read, or an X-register. A value of 0 in this bit, yielding Ix81 values from $08xxxx to $18xxxx, specifies a Y-register; a value of 1, yielding Ix81 values from $48xxxx to $58xxxx, specifies an X-register.

The following table shows Ix81 values for parallel data read through an ACC-14 board. All ACC-14 registers are a Y-addresses.

Register	Ix81	Register	Ix81
1st ACC-14D/V Port A	$xxFFD0	4th ACC-14D/V Port A	$xxFFE8
1st ACC-14D/V Port B	$xxFFD1	4th ACC-14D/V Port B	$xxFFE9
2nd ACC-14D/V Port A	$xxFFD8	5th ACC-14D/V Port A	$xxFFF0
2nd ACC-14D/V Port B	$xxFFD9	5th ACC-14D/V Port B	$xxFFF1
3rd ACC-14D/V Port A	$xxFFE0	6th ACC-14D/V Port A	$xxFFF8
3rd ACC-14D/V Port B	$xxFFE1	6th ACC-14D/V Port B	$xxFFF9
xx' represent the first two digits, which control bit width. They can take values from $08 to $18.			

For the ACC-8D Opt. 9 Yaskawa Absolute Encoder Converter, PMAC's 24-bit encoder phase position register, an X-register, is read, so Ix81 is set to $58xxxx ($180000 + $400000).

The following table shows Ix81 values for a Yaskawa absolute encoder connected through an ACC-8D Option 9 to each PMAC(1) encoder channel:

Channel	Ix81	Channel	Ix81
1	$58C001	9	$58C021
2	$58C005	10	$58C025
3	$58C009	11	$58C029
4	$58C00D	12	$58C02D
5	$58C011	13	$58C031
6	$58C015	14	$58C035
7	$58C019	15	$58C039
8	$58C01D	16	$58C03D
Channels 9 – 16 are on an ACC-24P/V board			

The following table shows Ix81 values for a Yaskawa absolute encoder connected through an ACC-8D Option 9 to each PMAC2 encoder channel:

Channel	Ix81	Channel	Ix81
1	$58C001	9	$58C041
2	$58C009	10	$58C049
3	$58C011	11	$58C051
4	$58C019	12	$58C059
5	$58C021	13	$58C061
6	$58C029	14	$58C069
7	$58C031	15	$58C071
8	$58C039	16	$58C079
Channels 9 – 16 are on an ACC-24P/V2 board			

For the ACC-49 Sanyo Absolute Encoder Converter, the encoder provides a 13-bit value within one motor revolution, and the data is read from a Y-register, so Ix81 is set to $0Dxxxx.

The following table lists the possible values of Ix81 for the ACC-49:

Enc. # on Board	Ix81 for E1 ON	Ix81 for E2 ON	Ix81 for E3 ON	Enc. # on Board	Ix81 for E4 ON	Ix81 for E5 ON	Ix81 for E6 ON
Enc. 1	$0DFFD0	$0DFFD8	$0DFFE0	Enc. 3	$0DFFE8	$0DFFF0	$0DFFF8
Enc. 2	$0DFFD4	$0DFFDC	$0DFFE4	Enc. 4	$0DFFEC	$0DFFF4	$0DFFFC

MACRO R/D Read: If Ix81 contains a value of $73000n, Motor x will read the absolute phase position from an ACC-8D Opt. 7 Resolver-to-Digital Converter through a MACRO Station or compatible device.

In this mode, the last hex digit 'n' of Ix81 specifies the MACRO node number. MACRO Station setup variable MI11x for the matching node must be set to read the R/D converter.

MACRO Parallel Read: If Ix81 contains a value of $74000n, Motor x will read the absolute phase position from a parallel data source through a MACRO Station or compatible device.

In this mode, the last hex digit 'n' of Ix81 specifies the MACRO node number. MACRO Station setup variable MI11x for the matching node must be set to read the parallel data source.

Hall Sensor Read: If Ix81 contains a value from $80xxxx to $FFxxxx (bit 23 if Ix81 set to 1), Motor x will read bits 20 through 22 of the PMAC memory or I/O register at the address specified by the low sixteen bits (last 4 hex digits 'xxxx') of Ix81. It will expect these three bits to be encoded as the U, V, and W "hall-effect" commutation signals with 120^{o}e spacing for the absolute power-on phase position. In this mode, the address specified in Ix81 is usually that of a flag register.

If the flag register is in a PMAC(1) or PMAC(1)-style ACC-24P, the flag inputs for bits 20, 21, and 22, representing W, V, and U, are +LIMn, -LIMn, and HMFLn, respectively. In a typical application, Ix81 specifies that these inputs are used from the "spare" flag register matching the second DAC channel used for commutation.

The following table shows the Ix81 settings for the flag registers in even-numbered channels of a PMAC(1) and a PMAC(1)-style ACC-24P that are typically used for hall commutation sensor inputs:

Channel	Ix81	Channel	Ix81
2	$xxC004	10	$xxC024
4	$xxC00C	12	$xxC02C
6	$xxC014	14	$xxC034
8	$xxC01C	16	$xxC03C
The proper value of 'xx' depends on the offset and direction sense of the hall sensors.			

If the flag register is in a PMAC2-style Servo IC, the input flags for bits 20, 21, and 22, representing W, V, and U, are CHWn, CHVn, and CHUn, respectively. In a typical application, these inputs are used from the same flag register addressed by Ix25 for the main flags.

The following table shows the Ix81 settings for the flag registers in channels of a PMAC2 that are typically used for hall commutation sensor inputs:

Channel	Ix81	Channel	Ix81
1	$xxC000	5	$xxC020
2	$xxC008	6	$xxC028
3	$xxC010	7	$xxC030
4	$xxC018	8	$xxC038
The proper value of 'xx' depends on the offset and direction sense of the hall sensors.			

In this mode, bit 22 of Ix81 allows for reversal of the sense of the hall-effect sensors. If W (bit 20 of the register; HMFLn or CHWn) leads V (bit 21; -LIMn or CHVn), and V leads U (bit 22; +LIMn or CHUn) as the commutation cycle counts up, then bit 22 of Ix81 should be set to 0. If U leads V and V leads W as the commutation cycle counts up, then bit 22 of Ix81 should be set to 1.

In this mode, bits 16 to 21 of Ix81 together form an offset value from 0 to 63 representing the difference between PMAC's commutation cycle zero and the hall-effect sensor zero position, which is defined as the transition of the V signal when U is low. This offset has units of 1/64 of a commutation cycle, or 5.625°e. Typically, one of the transitions will be at PMAC's commutation zero point, so the desired offset values will be 0°, 60°, 120°, 180°, 240°, and 300°, approximated by values of 0, 11($0B), 21($15), 32($20), 43($2B), and 53($35).

This operation can handle hall-effect sensors separated by 120°e. The following table gives the Ix81 settings for bits 16 to 23 for the most common cases of hall-effect settings as they relate to the PMAC commutation cycle.

0 to 60 deg	60 to 120deg	120 to 180 deg	180 to -120 deg	-120 to -60 deg	-60 to 0 deg	Ix81
011	010	110	100	101	001	$80xxxx
001	011	010	110	100	101	$8Bxxxx
101	001	011	010	110	100	$95xxxx
100	101	001	011	010	110	$A0xxxx
110	100	101	001	011	010	$ABxxxx
010	110	100	101	001	011	$B5xxxx
001	101	100	110	010	011	$C0xxxx
011	001	101	100	110	010	$CBxxxx
010	011	001	101	100	110	$D5xxxx
110	010	011	001	101	100	$E0xxxx
100	110	010	011	001	101	$EBxxxx
101	100	110	010	011	001	$F5xxxx
Note that '000' and '111' are invalid readings.						

Example Motor 1 has a single resolver at location 0 of an ACC-8D Opt.7 R/D converter board at multiplex address 0; no phasing search is permitted, but a homing search is required: I181=$000100 ($100=256dec, representing multiplex address 0), I110=0.

Motor 2 has a single resolver at location 6 of an ACC-8D Opt 7 board at multiplex address 4; no phasing search is permitted, but a homing search is required: I281=$060004; I210=0.

Motor 3 has a double geared resolver at locations 2 and 3 of an ACC-8D Opt 7 board at multiplex address 6, with a 10:1 gear ratio between them; no phasing search or homing search is permitted: I381=$020006; I310=$020006; I93=10

Motor 4 has a 20-bit single-turn absolute encoder at Port A of the first ACC-14 (address Y:$FFD0): I481=$14FFD0 ($14=20dec)

Motor 5 is a brush motor with a double geared resolver at locations 0 and 1 of an ACC-8D Opt 7 board at multiplex address 2; no homing search is permitted: I581=0 (no phasing required); I510=$000002

Motor 6 uses hall-effect sensors wired into the flags on Channel 12 for power-up phase referencing. The zero point of the hall effect is at 60°e, and the direction is "standard", not reversed. I610= $8BC034.

See Also Phasing Referenced to Absolute Sensor (Setting Up PMAC Commutation)
I-Variables I8x, I9x, Ix03, Ix10, Ix75, Ix83
ACC-8D Option 7 (R/D Converter) Manual

Ix82 Current loop Feedback Address {PMAC2 only}

Range Legal PMAC Y addresses

Units Legal PMAC Y addresses

Default 0

Remarks Ix82 tells PMAC2 which addresses to read to get its current feedback values for Motor x if PMAC2 is closing the current loop for this motor. PMAC must be performing the commutation for the motor (Ix01=1) if it is to close the current loop as well.

A zero value for Ix82 tells PMAC2 not to close the current loop for this motor. In this case, PMAC either outputs one velocity or torque command value (Ix01=0), or two phase-current command values (Ix01=1), usually represented as analog voltages.

A non-zero value for Ix82 automatically triggers current loop execution in the phase interrupt, using the current value(s) found in the register(s) specified by Ix82. Typically these registers are analog-to-digital converter (ADC) registers in the PMAC2 ASIC, or MACRO feedback registers containing copies of ADC registers in a MACRO Station

When Ix01 is set to 1, PMAC2 performs the phase commutation for this motor, computing two phase current commands based on the position/velocity servo command and the magnetization current value. If Ix82>0, these commands are compared to the two actual current values read from the address specified by Ix82, and the next *lower* address. It executes a PI filter on the current loops and outputs three voltage command values to the address specified by Ix02 and the next two higher addresses. These are typically the PWM commands for the three half-bridges of a brushless motor power stage.

When the digital current loop is used for drives connected directly to the PMAC2, the typical values for Ix82 are:

Channel	Ix82	Channel	Ix82
ADC1A & B	$C006	ACDC9A & B	$C046
ADC2A & B	$C00E	ACDC10A & B	$C04E
ADC3A & B	$C016	ACDC11A & B	$C056
ADC4A & B	$C01E	ACDC12A & B	$C05E
ADC5A & B	$C026	ACDC13A & B	$C066
ADC6A & B	$C02E	ACDC14A & B	$C06E
ADC7A & B	$C036	ACDC15A & B	$C076
ADC8A & B	$C03E	ACDC16A & B	$C07E
Channels 9 – 16 are present on an ACC-24P/V2 board			

When the digital current loop is used for drives connected to the PMAC2 through a MACRO station, the typical values for Ix82 are:

Node/Register	Ix82	Node/Register	Ix82
Node 0/Reg 1 & 2	$C0A2	Node 8/Reg 1 & 2	$C0B2
Node 1/Reg 1 & 2	$C0A6	Node 9/Reg 1 & 2	$C0B6
Node 4/Reg 1 & 2	$C0AA	Node 12/Reg 1 & 2	$C0BA
Node 5/Reg 1 & 2	$C0AE	Node 13/Reg 1 & 2	$C0BE

If Ix82>0, the following variables must be set properly for correct operation of the digital current loop:

Ix61: Current-Loop Integral Gain
Ix62: Current-Loop Forward-Path Proportional Gain
Ix66: PWM Scale Factor
Ix72: Commutation Phase Angle
Ix76: Current-Loop Back-Path Proportional Gain
Ix84: Current-Loop Feedback Mask Word

Ix83 Motor x Ongoing Phasing Position Address

Range Legal PMAC X and Y addresses

Units Legal PMAC X and Y addresses

Default

Variable	PMAC(1)	PMAC2	PMAC2 Ultralite
I183	$C001	$C001	$8C0A0
I283	$C009	$C009	$8C0A4
I383	$C011	$C011	$8C0A8
I483	$C019	$C019	$8C0AC
I583	$C021	$C021	$8C0B0
I683	$C029	$C029	$8C0B4
I783	$C031	$C031	$8C0B8
I883	$C039	$C039	$8C0BC

Remarks For a motor commutated by PMAC2, this parameter tells PMAC2 where to read its commutation (phasing) position information for Motor x every commutation cycle. This can be a different address from that used for power-on/reset phasing position, which is determined by Ix81.

Bits 0 to 15 of Ix83 contain the 16-bit address of the register to be read. Bit 19 of Ix83 tells whether the register has a X address or a Y address; a 0 value specifies X, and a 1 value (which makes the hexadecimal digit have an 8 value) specifies Y.

For PMAC(1) and PMAC2 boards with on-board encoder circuitry, Ix83 typically contains the address of the phase position encoder register for encoder *x*; this is the default. Since these registers have X addresses, bit 19 is 0.

On PMAC(1) boards, because two channels are required for commutation output, usually only the odd-numbered channels are used for commutation feedback. This is reflected in the defaults.

The following table provides the Ix83 values for all of the possible phase-position registers in PMAC(1) system:

Channel	Ix83	Channel	Ix83
1	$C001	9	$C021
2	$C005	10	$C025
3	$C009	11	$C029
4	$C00D	12	$C02D
5	$C011	13	$C031
6	$C015	14	$C035
7	$C019	15	$C039
8	$C01D	16	$C03D
Channels 9 – 16 are present on an ACC-24P/V board			

On PMAC2 boards, commutation requires only one channel, so any channel can be used for commutation feedback. The following table provides the Ix83 values for all of the phase-position registers in a PMAC2 system:

Channel	Ix83	Channel	Ix83
1	$C001	9	$C041
2	$C009	10	$C049
3	$C011	11	$C051
4	$C019	12	$C059
5	$C021	13	$C061
6	$C029	14	$C069
7	$C031	15	$C071
8	$C039	16	$C079
Channels 9 – 16 are present on an ACC-24P/V2 board			

For PMAC2 Ultralite boards, Ix83 typically contains the address of a MACRO node's position feedback register; this is the default. Since PMAC2 can only commutate over MACRO using nodes with 'Y' addresses, bit 19 must be set to 1 in these cases. The following table shows Ix83 values for all of the MACRO servo nodes:

Node	Ix83	Channel	Ix83
0	$8C0A0	8	$8C0B0
1	$8C0A4	9	$8C0B4
4	$8C0A8	12	$8C0B8
5	$8C0AC	13	$8C0BC

If the motor is performing open-loop microstepping control inside PMAC (Ix01=1, bit 16 of Ix02=1), this parameter must contain the address of the motor's "phase advance " register (X:$0042, X:$007E, etc.) instead of an encoder register.

See Also I-variables Ix01, Ix02, Ix03, Ix04, Ix81
Setting Up PMAC Commutation
I/O-Memory Map registers X:$C001, X:$C005, etc., X:$0042, X:$007E, etc.

Ix84 Current-Loop Feedback Mask Word {PMAC2 only}

Range $000000 .. $FFFFFF

Units Bit mask

Default $FFF000 (12-bit ADCs)

Remarks Ix84 tells PMAC2 what bits of the 24-bit current feedback word(s) to use as actual the actual current value in the current loop equations when it is closing the current loops for a direct-PWM "power-block" amplifier. It is only used if Ix82>0, enabling current loop closure in the PMAC2.

PMAC2 supports interface to serial analog-to-digital converters of many resolutions through its "DSPGATE1" ASIC. The data is received in 18-bit shift registers in the ASIC, which are read as the high end of a 24-bit word, with the number "left-justified" to the most significant bit.

Ix84 specifies a 24-bit mask word that is combined with the feedback word through a logical AND operation to produce the value that is used in the current loop equations. There should be a 1 in every bit that is used, and a 0 in every bit that is not. Since the data is left justified, Ix84 should start with 1s and end with 0s. Usually Ix84 is represented as a hexadecimal number, with 4 bits per digit, and a total of six digits

Some direct-PWM amplifiers will transmit status and fault information on the end of the serial data stream for the ADC, and it is important to mask out these values from the current loop equations.

Example For a 10-bit ADC: Ix84=$FFC000
For a 12-bit ADC: Ix84=$FFF000
For a 16-bit ADC: Ix84=$FFFF00

Further Motor I-Variables

Ix85 Motor x Backlash Take-up Rate

Range 0 .. 8,388,607

Units (1/16 Counts) / Background Cycle

Default 0

Remarks Ix85 determines how fast backlash is "taken up" on direction reversal. The size of the backlash is determined by Ix86, and possibly the backlash compensation table for the motor. PMAC will "take up" the backlash at the Ix85 rate whenever the commanded or Master Handwheel position for the motor reverses more than 4 encoder counts. If Ix85 is zero, backlash is effectively disabled. Ix85 is usually set as high as possible without creating dynamic problems.

Variable I99, Backlash Hysteresis, determines the amount of reversal in desired position that is required before backlash will start to be introduced or removed.

See Also I-variables I99, Ix64, Ix65, Ix68, Ix86
On-line commands **`DEFINE BLCOMP`**, **`DELETE BLCOMP`**
Backlash Compensation (Setting Up a Motor)

Ix86 Motor x Backlash Size

Range -8,388,608 .. 8,388,607

Units 1/16 Count

Default 0

Remarks Ix86 allows PMAC to compensate for backlash in the motor's coupling by adding or subtracting (depending on the new direction) the amount specified in the parameter to the commanded position on direction reversals (this offset will not appear when position is queried or displayed). A value of zero means no backlash. Negative values of Ix86 can be useful if the motor is slaved to another motor that has more backlash than the slave.

The rate at which this backlash is added or subtracted (taken up) is determined by Ix85. Variable I99, Backlash Hysteresis, determines the amount of reversal in desired position that is required before backlash will start to be introduced or removed.

If backlash tables are used, Ix86 represents the backlash at motor zero position; values in the table should represent the difference between the backlash at a given position and Ix86.

Note:

The units of this parameter are 1/16 of a count so the value should be 16 times the number of counts of backlash compensation desired.

Example If you find that you have a backlash on reversal of motor direction of 7.5 encoder counts, you would set Ix86 to 7.5 * 16 = 120.

See Also I-variables I99, Ix64, Ix65, Ix68, Ix85
On-line commands **DEFINE BLCOMP**, **DELETE BLCOMP**
Backlash Compensation (Setting Up a Motor)

Coordinate System x I-Variables

x = Coordinate System Number (&x, x = 1 to 8)

Ix87 Coordinate System x Default Program Acceleration Time

Range 0 .. 8,388,607

Units msec

Default 0 (so Ix88 controls)

Remarks Ix87 sets the default time for commanded acceleration for programmed blended **LINEAR** and **CIRCLE** mode moves in Coordinate System x. It also provides the default segment time for **SPLINE** mode moves. The first use of a **TA** statement in a program overrides this value.

Note:

Even though this parameter makes is possible not to specify acceleration time in the motion program, you are strongly encouraged to use **TA** in the program and not rely on this parameter, unless you must keep to a syntax standard that does not support this (e.g. RS-274 "G-Codes "). Specifying acceleration time in the program along with speed and move modes makes it much easier for later debugging.

If the specified S-curve time (see Ix88, below) is greater than half the specified acceleration time, the time used for commanded acceleration in blended moves will be twice the specified S-curve time.

The acceleration time is also the minimum time for a blended move; if the distance on a feedrate-specified (F) move is so short that the calculated move time is less than the acceleration time, or the time of a time-specified (TM) move is less than the acceleration time, the move will be done in the acceleration time instead. This will slow down the move.

Note:

The acceleration time will be extended automatically when any motor in the coordinate system is asked to exceed its maximum acceleration rate (Ix17) for a programmed LINEAR-mode move with I13=0 (no move segmentation).

Note:

Make sure that the specified acceleration time (Ix87 or 2*Ix88) is greater than zero, even if you are planning to rely on the maximum acceleration rate parameters. A specified acceleration time of zero will cause a divide-by-zero error. The minimum specified time should be Ix87=1, Ix88=0.

See Also Acceleration Limits (Making Your Application Safe)
I-variables I13, Ix17, Ix88
Program Commands **TA**, **TS**

Ix88 Coordinate System x Default Program S-Curve Time

Range 0 .. 8,388,607

Units msec

Default 50

Remarks Ix88 set the default time in each half of the S in S-curve acceleration for programmed blended **LINEAR** and **CIRCLE** mode moves in Coordinate System x. It does not affect **SPLINE**, **PVT**, or **RAPID** mode moves. The first use of a **TS** statement in a program overrides this value.

Note:

Even though this parameter makes is possible not to specify acceleration time in the motion program, you are strongly encouraged to use **TS** in the program and not rely on this parameter, unless you must keep to a syntax standard that does not support this (e.g. RS-274 G-Codes). Specifying acceleration time in the program along with speed and move modes makes it much easier for later debugging.

If Ix88 is zero, the acceleration is constant throughout the Ix87 time and the velocity profile is trapezoidal. If Ix88 is greater than zero, the acceleration will start at zero and linearly increase through Ix88 time, then stay constant (for time TC) until Ix87-Ix88 time, and linearly decrease to zero at Ix87 time (that is Ix87=2*Ix88 - TC). If Ix88 is equal to Ix87/2, the entire acceleration will be spec in S-curve form (Ix88 values greater than Ix87/2 override the Ix87 value; total acceleration time will be 2*Ix88).

Note:

The acceleration time will be extended automatically when any motor in the coordinate system is asked to exceed its maximum acceleration rate (Ix17) for a programmed LINEAR-mode move with I13=0 (no move segmentation).

Make sure the specified acceleration time (TA or 2*TS) is greater than zero, even if you are planning to rely on the maximum acceleration rate parameters (Ix17). A specified acceleration time of zero will cause a divide-by-zero error. The minimum specified time should be **TA1 TS0**.

See Also Acceleration Limits (Making Your Application Safe)
I-variables I13, Ix17, Ix87
Program Commands **TA**, **TS**

Ix89 Coordinate System x Default Program Feedrate/Move Time

Range Positive floating point

Units (user position units)/(feedrate time units) for feedrate
msec for move time

Default 1000.0

Remarks Ix89 sets the default feedrate (commanded speed) for programmed **LINEAR** and **CIRCLE** mode moves in Coordinate System x. The first use of an **F** or **TM** statement in a motion program overrides this value. The velocity units are defined by the position and time units, as defined by axis definition statements and Ix90. After power-up/reset, the coordinate system is in feedrate mode, not move time mode.

> ***Note:***
>
> You are strongly encouraged *not* to rely on this parameter and to declare your feedrate in the program. This will keep your move parameters with your move commands, lessening the chances of future errors, and making debugging easier.

When polled, Ix89 will report the value from the most recently executed **F** or **TM** command in that coordinate system.

See Also Axis Definition Statements (Setting Up a Coordinate System)
LINEAR and **CIRCLE** mode blended moves (Writing a Motion Program)
I-variables Ix87, Ix88, Ix90
Program commands **F**, **TM**

Ix90 Coordinate System x Feedrate Time Units

Range positive floating point

Units msec

Default 1000.0 (velocity time units are seconds)

Remarks Ix90 defines the time units used in commanded velocities (feedrates) in motion programs executed by Coordinate System x. Velocity units are comprised of length units divided by time units. The length units are determined in the axis definition statements for the coordinate system. Ix90 sets the time units. Ix90 itself has units of milliseconds, so if Ix90 is 60,000, the time units are 60,000 milliseconds, or minutes. The default value of Ix90 is 1000 msec, specifying velocity time units of seconds.

This affects two types of motion program values: **F** values (feedrate) for **LINEAR**- and **CIRCLE**-mode moves; and the velocities in the actual move commands for **PVT**-mode moves.

Example If position units have been set as centimeters by the axis definition statements, and it is desired that feedrate values be specified in cm/sec, this parameter would be set to 1000.0 (time units = sec).

If position units have been set as degrees by the axis definition statements, and it is desired that feedrate values be specified in deg/min, this parameter would be set to 60,000.0 (time units = minutes).

If a spindle is rotating at 4800 rpm, with a linear axis specified in inches, and it is desired that linear speed be specified in inches per spindle revolution, Ix90 would be set to 12.5 ([1 min/4800 rev] * [60,000 msec/ min] = 12.5 msec/rev).

See Also Axis Definition Statements (Setting Up a Coordinate System)
Velocity-Specified Moves, PVT-Mode Moves (Writing a Motion Program)
Motion program commands **F{data}**, **{axis}{data}:{data}**.

Ix91 Coordinate System x Default Working Program Number

Range 0 .. 32,767

Units Motion Program Numbers

Default 0

Remarks Ix91 tells PMAC which motion program to run in this coordinate system when commanded to run from the control-panel input (START/ or STEP/ line taken low). It performs the same function for a hardware run command as the **B** command does for a software run command (**R**). It is intended primarily for stand-alone PMAC applications. The first use of a **B** command from a host computer for this coordinate system overrides this parameter.

See Also Control-Panel Port Inputs (Connecting PMAC to the Machine)
On-line commands **B{constant}**, **R**, **S**.

Ix92 Coordinate System x Move Blend Disable

Range 0 .. 1

Units None

Default 0

Remarks Ix92 controls whether Coordinate System x automatically blends moves together or not. If Ix92 set to 0, programmed blended moves – **LINEAR**, **SPLINE**, and **CIRCLE**-mode – are blended together with no intervening stop. Upcoming moves are calculated during the current moves.

If Ix92 is set to 1, there is a brief stop in between each programmed move (it effectively adds a **DWELL 0** command), during which the next move is calculated. The calculation time for the next move is determined by I11.

This parameter is only acted upon when the **R** or **S** command is given to start program execution. To change the mode of operation while the program is running the "continuous motion request" coordinate system status bit (bit 4 of X:$0818 etc.) must be changed. The polarity of this bit is opposite that of Ix92.

See Also **LINEAR**- and **CIRCLE**-Mode Blended Moves (Writing a Motion Program)
How PMAC Executes a Motion Program (Writing a Motion Program)
I-variable I11

Ix93 Coordinate System x Time Base Control Register Address

Range Legal PMAC X addresses

Units Legal PMAC addresses

Default

Variable	Hex	Decimal	Register
I193	$0806	2054	C.S.1 '%' cmd reg
I293	$08C6	2246	C.S.2 '%' cmd reg
I393	$0986	2438	C.S.3 '%' cmd reg
I493	$0A46	2630	C.S.4 '%' cmd reg
I593	$0B06	2822	C.S.5 '%' cmd reg
I693	$0BC6	3014	C.S.6 '%' cmd reg
I793	$0C86	3206	C.S.7 '%' cmd reg
I893	$0D46	3398	C.S.8 '%' cmd reg

Remarks Ix93 tells Coordinate System x where to look for its time base control (feedrate override) information by specifying the address of the register that will be used. The default value of this parameter for each coordinate system (see above) specifies the register that responds to on-line commands. If the time base is left alone, or is under host or programmatic control, this parameter should be left at the default.

Alternatively, if the time base is controlled externally from a frequency or voltage, the register containing the time-base information will almost always be in the conversion table (which starts at address $720 [1824 decimal]). With the default conversion table, there is a time-base register at $0729 (1833) related to the frequency into the Encoder 4 counter. This frequency can be controlled by an input voltage on the WIPER pin of the Control Panel Port if jumpers E72 and E73 are ON. If another register is to be used for the time base, it must have the units of I10 so that 8388608 (2^{23}) indicates 1 msec between servo interrupts. See instructions for using an external time base, under Synchronizing PMAC to External Events.

Note:

Ix93 contains the *address* of the register that holds the time-base value (it is a pointer to that register). Ix93 does not contain the time-base value itself.

See Also Time-Base Control (Synchronizing PMAC to External Events)
Control Panel Port Inputs (Connecting PMAC to the Machine)
I-variables I10, Ix93, Ix95
On-line commands `%`, `%{constant}`.
Jumpers E72, E73

Ix94 Coordinate System x Time Base Slew Rate

Range 0 .. 8,388,607

Units 2^{-23}msec/ servo cycle

Default 1644

Remarks Ix94 controls the rate of change of the coordinate system's time base. It effectively works in two slightly different ways, depending on the source of the time base information. If the source of the time base is the "%" command register, then Ix94 defines the rate at which the " (actual time base) value will slew to a newly commanded value. If the rate is too high, and the % value is changed while axes in the coordinate system are moving, there will be a virtual step change in velocity. For these type of applications, Ix94 is set relatively low (often 1000 to 5000) to provide smooth changes.

Note:

The default Ix94 value of 1644, when used on a card set up with the default servo cycle time of 442 µsec, provides a transition time between %0 and %100 of one second.

If there is a hardware source (as defined by Ix93), the commanded time-base value changes every servo cycle, and the rate of change of the commanded value is typically limited by hardware considerations (e.g. inertia). In this case, Ix94 effectively defines the maximum rate at which the % value can slew to the new hardware-determined value, and the actual rate of change is determined by the hardware. If you wish to keep synchronous to a hardware input frequency, as in a position-lock cam, Ix94 should be set high enough that the limit is never activated. However, following motion can be smoothed significantly with a lower limit if total synchronicity is not required.

See Also Time-Base Control (Synchronizing PMAC to External Events)
I-variables I10, Ix93, Ix95
On-line commands **%{constant}**, **%**

Ix95 Coordinate System x Feed Hold Slew Rate

Range 0 .. 8,388,607

Units 2^{-23}msec/servo cycle

Default 1644

Remarks Ix95 controls the rate at which the axes of the coordinate system stop if a feed hold command (**H**) is given, and the rate at which they start up again on a succeeding run command (**R** or **S**). A feed hold command is equivalent to a **%0** command except that it uses Ix95 for its slew rate instead of Ix94. Having separate slew parameters for normal time-base control and for feed hold commands allows both responsive ongoing time-base control (Ix94 relatively high) and well-controlled holds (Ix95 relatively low).

> ***Note:***
>
> The default Ix95 value of 1644, when used on a card set up with the default servo cycle time of 442 μsec, provides a transition time between %100 and %0 (feed hold) of one second.

See Also Stop Commands (Making Your Application Safe)
Time-Base Control (Synchronizing PMAC to External Events)
I-variables I10, Ix93, Ix94
On-line commands **H**, **<CONTROL-O>**, **R**, **<CONTROL-R>**, **S**, **<CONTROL-S>**, **%**.

Ix96 Coordinate System x Circle Error Limit

Range positive floating point

Units User length units

Default 0 (function disabled)

Remarks In a circular arc move, a move distance that is more than twice the specified radius will cause a computation error because a proper path cannot be found. Sometimes, due to round-off errors, a distance slightly larger than twice the radius is given (for a half-circle move), and it is desired that this not create an error condition.

Ix96 allows the user to set an error limit on the amount the arc move distance is greater than twice the specified radius. If the move distance is greater than 2R, but by less than this limit, the move is done in a spiral fashion to the endpoint, and no error condition is generated. If the distance error is greater than this limit, a run-time error will be generated, and the program will stop. If this variable is set to 0 the error generation is disabled and any move distance greater than 2R is done in a spiral fashion to the endpoint.

If the circular move is specified with an IJK center-vector instead of an R radius, Ix96 is not used.

Example Given the program segment

```
INC CIRCLE1 F2
X7.072 Y7.072 R5
```

technically no circular arc path can be found, because the distance is SQRT(7.072^2+7.072^2) = 10.003, which is greater than twice the radius of 5. However as long as Ix96 is greater than 0.003, PMAC will create a near-circular path to the end point.

See Also Circular Blended Moves (Writing a Motion Program)
Program commands **CIRCLE1**, **CIRCLE2**, **{axis}{data}{vector}{data}**

Ix97 (Reserved for Future Use)

Ix98 Coordinate System x Maximum Feedrate

Range Non-negative floating-point

Units User axis length/angle units per Ix90 milliseconds

Default 0

Remarks Ix98 permits a maximum feedrate to be set for a coordinate system, preventing a program from accidentally exceeding a specified value. If Ix98 is greater than 0, PMAC will compare each commanded vector feedrate value from an **F** command in a motion program to Ix98. If the commanded feedrate is greater than Ix98, it will use Ix98 instead.

If Ix98 is set to 0, PMAC will not check the programmed feedrate value against a limit.

Ix99 (Reserved for Future Use)

PMAC(1) Encoder/Flag Setup I-Variables

One PMAC can have up to 16 incremental encoder channels – four per gate array IC. Each encoder and its related flags and registers are set up using (up to) 5 I-variables. The encoders and their flags are numbered 1 to 16, matching the numbers of their pinouts (e.g. CHA1, CHB1, and CHC1 belong to encoder 1.) The encoder I-variables are assigned to the different encoders as follows:

I900 - I904 – Encoder 1
I905 - I909 – Encoder 2
I910 - I914 – Encoder 3
I915 - I919 – Encoder 4
...
I970 - I974 – Encoder 15
I975 - I979 – Encoder 16

An encoder is assigned to a motor for position, velocity (feedback), handwheel (master), or feedpot (frequency control) by using the appropriate motor I-variables (see above).

I900, I905, ..., I975 Encoder n Decode Control "Encoder I-Variable 0" {PMAC(1) Only}

Range 0 .. 15

Units none

Default 7

Remarks

WARNING:

Changing the direction sense of the encoder decode for a motor that is servoing properly will result in unstable positive feedback and a dangerous runaway condition in the absence of other changes (for motors not commutated by PMAC from the same encoder). The output polarity must be changed as well to re-establish polarity match for stable negative feedback.

This parameter controls how the input signal for Encoder n is decoded into counts. As such, this defines the sign and magnitude of a count.

The following settings may be used to decode an input signal.

Setting	Meaning
0	pulse and direction CW
1	x1 quadrature decode CW
2	x2 quadrature decode CW
3	x4 quadrature decode CW
4	pulse and direction CCW
5	x1 quadrature decode CCW
6	x2 quadrature decode CCW
7	x4 quadrature decode CCW

In any of the quadrature decode modes, PMAC is expecting two input waveforms on CHAn and CHBn, each with approximately 50% duty cycle, and approximately one-quarter of a cycle out of phase with each other. "Times-one" (x1) decode provides one count per cycle; x2 provides two counts per cycle; and x4 provides four counts per cycle. The vast majority of users select x4 decode to get maximum resolution.

The "clockwise" (CW) and "counterclockwise" (CCW) options simply control which direction counts up. If you get the wrong direction sense, simply change to the other option (e.g. from 7 to 3 or vice versa).

In the pulse-and-direction decode modes, PMAC is expecting the pulse train on CHAn, and the direction (sign) signal on CHBn. If the signal is unidirectional, the CHBn input can be tied high (to +5V) or low (to GND), or, if set up by E18-E21, E24-E27 for single-ended (non-differential) input, left to float high.

Any spare encoder counters may be used as fast and accurate timers by setting this parameter in the 8 to 15 range. In this range, any input signal is ignored. The following settings may be used in timer mode:

Setting	Meaning
8	Timer counting up at SCLK/10
9	Timer counting up at SCLK/10
10	Timer counting up at SCLK/5
11	Timer counting up at SCLK/2.5
12	Timer counting down at SCLK/10
13	Timer counting down at SCLK/10
14	Timer counting down at SCLK/5
15	Timer counting down at SCLK/2.5

These timers are particularly useful when the related capture and compare registers are utilized for precise event marking and control, including triggered time base. The SLCK frequency is determined by the crystal clock frequency and E34-E38.

See Also Triggered Time Base (Synchronizing PMAC to External Events)
I-variables Ix03-Ix05, Ix93
Jumpers E18-E21, E24-E27, E34-E38.

I901, I906, ..., I976 Encoder n Filter Disable "Encoder I-Variable 1" {PMAC(1) Only}

Range 0 .. 1

Units none

Default 0

Remarks This parameter controls whether the encoder channel enables or disables its digital delay filter. The options are:

0 = Encoder n digital delay filter enabled

1 = Encoder n digital delay filter disabled (bypassed)

The filter is a 3-stage digital delay filter with best-2-of-3 voting to help suppress noise spikes on the input lines. It does introduce a small delay into the signal, which can be unacceptable if the motor is using interpolated sub-count parallel data input, because of loss of synchronization between the quadrature and parallel data signals.

Note:

Generally, the only people to disable this filter are those using the special interpolated parallel data format. These people should disable the filters both on the encoder for their quadrature signals and the encoder matching their parallel data input.

The sampling frequency for the filter is that of the SCLK signal, which is set by the master clock frequency and jumpers E34-E38. The higher the frequency of SCLK, the higher the possible count rate, but the narrower the pulse that can be filtered out. SCLK should be set to allow the maximum expected encoder frequency, but no faster, in order to provide the maximum noise protection.

See Also Digital Delay Filter (Connecting PMAC to the Machine)
Parallel Sub-Count Interpolation (Setting Up a Motor)
Jumpers E34-E38

I902, I907, ..., I977 Encoder n Position Capture Control "Encoder I-Variable 2" {PMAC(1) Only}

Range 0 .. 15

Units none

Default 1

Remarks This parameter determines which signal or combination of signals (and which polarity) triggers a position capture of the counter for encoder n. If a flag input (home, limit, or fault) is used, I903 (etc.) determines which flag. Proper setup of this variable is essential for a successful home search, which depends on the position-capture function.

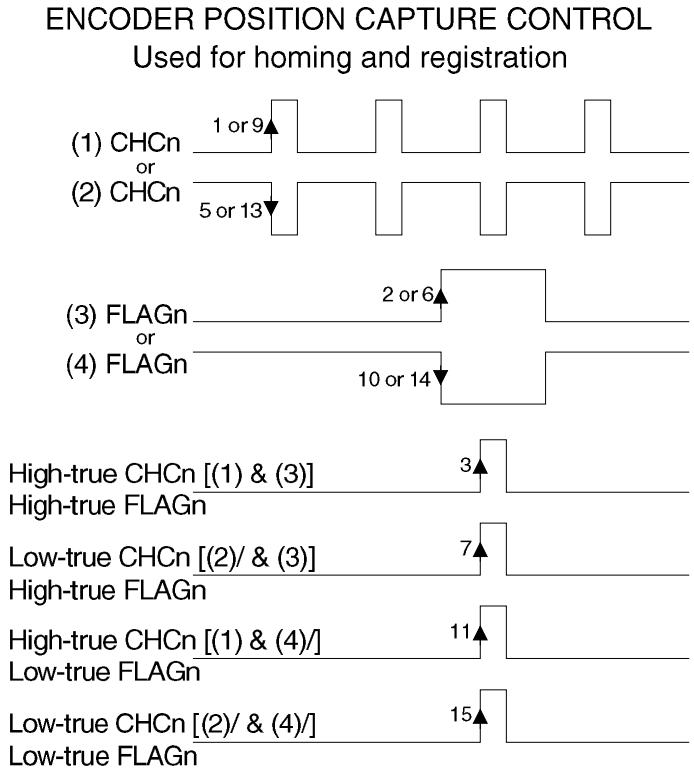
The following settings may be used:

Setting	Meaning
0	Software Control (armed)
1	Rising edge of CHCn (third channel)
2	Rising edge of Flag n (as set by Flag Select)
3	Rising edge of [CHCn AND Flag n] – Low true index, high true Flag
4	Software Control (triggered)
5	Falling edge of CHCn (third channel)
6	Rising edge of Flag n (as set by Flag Select)
7	Rising edge of [CHCn/ AND Flag n] – Low true index, high true Flag
8	Software Control (armed)
9	Rising edge of CHCn (third channel)
10	Falling edge of Flag n (as set by Flag Select)
11	Rising edge of [CHCn AND Flag n/] – High true index, low true Flag
12	Software Control (triggered)
13	Falling edge of CHCn (third channel)
14	Falling edge of Flag n (as set by Flag Select)
15	Rising edge of [CHCn/ AND Flag n/] – Low true index, low true Flag

Note that several of these values are redundant. To do a software-controlled position capture, preset this parameter to 0 or 8; when the parameter is then changed to 4 or 12, the capture is triggered (this is not of much practical use, but can be valuable for testing the capture function).

See Also Position Capture (Synchronizing PMAC to External Events)
Homing Moves (Basic Motor Moves)
I-variables Ix25, Encoder I-Variable 3

I902, I907, ... , I977



I903, I908, ..., I978 Encoder n Flag Select Control Encoder I-Variable 3 {PMAC(1) only}

Range 0 .. 3

Units None

Default 0

Remarks This parameter determines which of the Flag inputs will be used for position capture (if one is used – see I902 etc.):

Setting	Meaning
0	HMFLn (Home Flag n)
1	-LIMn (Positive Limit Signal n)
2	+LIMn (Negative Limit Signal n)
3	FAULTn (Amplifier Fault Signal n)

This parameter is typically set to zero, because in actual use, the +/-LIMn and FAULTn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. If you wish to capture on the +/-LIMn or FAULTn flags, you must either disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

Note:

The direction sense of the limit inputs is the opposite of what many people consider intuitive. That is, the +LIMn input, when taken high (opened), stops commanded motion in the negative direction; the -LIMn input, when taken high, stops commanded motion in the positive direction. It is important to confirm the direction sense of your limit inputs in actual operation.

See Also I-variables Ix25, I902
Position Capture (Synchronizing PMAC to External Events)
Homing Moves (Basic Motor Moves)

I904, I909, .., I979 – (Reserved for Future Use) {PMAC(1) only}

PMAC2 Encoder/Flag/Output Setup I-Variables

The DSPGATE1 Servo ICs of PMAC2 controllers have several setup variables. PMAC2 has I-variables for the important setup registers of 2 Servo ASICs comprising eight servo interface channels. It is possible to use two additional Servo ASICs on ACC-24P2 or ACC-51P boards, but these do not have I-variables assigned to their setup registers.

Global / Multi-Channel ASIC I-Variables

The I-variables I900 – I909 on a PMAC2 controller control the global setup registers of the two possible on-board Servo ASICs of the PMAC2. Several of these registers on the first Servo ASIC control important parameters for the whole PMAC2 system.

I900 MaxPhase and PWM 1-4 Frequency Control {PMAC2 only}

Range 0 .. 32767

Units MaxPhase Frequency = 117,964.8 kHz / [2*I900+3]
PWM Frequency = 117,964.8 kHz / [4*I900+6]

Default 6527
MaxPhase Frequency = 117,964.8 / 13057 = 9.0346 kHz
PWM Frequency = 117,964.8 / 26114 = 4.5173 kHz

Remarks I900 controls the maximum phase clock frequency for the PMAC2, and the PWM frequency for machine interface channels 1-4. It does this by setting the limits of the PWM up-down counter, which increments and decrements at the PWMCLK frequency of 117,964.8 kHz (117.9648 MHz).

The actual phase clock frequency is divided down from the maximum phase clock according to the setting of I901. On the falling edge of the phase clock, PMAC2 samples any serial analog-to-digital converters connected to its ASICs (as for phase current measurement), and interrupts the processor to start any necessary phase commutation and digital current-loop algorithms. Even if phasing and current-loop algorithms are not used, the MaxPhase and PhaseClock frequencies are important because the servo clock is derived from the phase clock.

The PWM frequency determines the actual switching frequency of amplifiers connected to any of PMAC2's first four machine interface channels with the direct PWM command. It is only important if the direct PWM command signal format is used.

The maximum value that can be written into the PWM command register without full saturation is I900+1 on the positive end, and -I900-2 on the negative end. Generally, the PWM scale factor Ix66 for Motor, which determines the maximum PWM command magnitude, is set to I900 + 10%.

To set I900 for a desired PWM frequency, the following formula can be used:

$$I900 = \frac{117{,}964.8(kHz)}{4 * PWM\_Freq(kHz)} - 1 \text{ (rounded down)}$$

To set I900 for a desired "maximum phase" clock frequency, the following formula can be used:

$$I900 = \frac{117{,}964.8(kHz)}{2 * MaxPhaseFreq(kHz)} - 1 \text{ (rounded down)}$$

To set a PWM frequency of 10 kHz and therefore a MaxPhase clock frequency of 20 kHz:

$$I900 = (117{,}964.8 \text{ kHz} / [4*10 \text{ kHz}]) - 1 = 2948$$

Example To set a PWM frequency of 7.5 kHz and therefore a MaxPhase clock frequency of 15 kHz:

*I900 = (117,964.8 kHz / [4*7.5 kHz]) - 1 = 3931*

See Also I901, I902, I905, I906, I992

I901 Phase Clock Frequency Control {PMAC2 only}

Range 0 .. 15

Units PHASE Clock Frequency = MaxPhase Frequency / (I901+1)

Default 0
PHASE Clock Frequency = 9.0346 kHz / 1 = 9.0346 kHz
(with default value of I900)

Remarks I901, in conjunction with I900, determines the frequency of the PHASE clock on PMAC2 (except for PMAC2 Ultralites, which use I992 and I997 for this). Each cycle of the PHASE clock, motor phase commutation and digital current-loop algorithms are performed for specified motors.

Specifically, I901 controls how many times the PHASE clock frequency is divided down from the "maximum phase" clock, whose frequency is set by I900. The PHASE clock frequency is equal to the "maximum phase" clock frequency divided by (I901+1). I901 has a range of 0 to 15, so the frequency division can be by a factor of 1 to 16. The equation for I901 is:

$$I901 = \frac{MaxPhaseFreq(kHz)}{PhaseFreq(kHz)} - 1$$

The ratio of MaxPhase Freq. to PHASE Clock Freq. must be an integer.

Note:

If jumper E1 is ON, PMAC2 gets its PHASE clock signal externally from a serial-port input, and I901 is not used.

Note:

If the phase clock frequency is set too high, lower priority tasks such as communications can be starved for time. If the background tasks are completely starved, the watchdog timer will trip, shutting down the board. If a normal reset of the board does not re-establish a state where the watchdog timer has not tripped and communications works well, it will be necessary to re-initialize the board by powering up with the E3 re-initialization jumper on. This restores default settings, so communication is possible, and I900 and I901 can be set to supportable values.

Example With a 20 kHz MaxPhase Clock frequency established by I900, and a desired 6.67 kHz PHASE clock frequency, the ratio between MaxPhase and PHASE is 3:
I901 = (20 / 6.67) - 1 = 3 -1 = 2

See Also I900, I902, I997

I902 Servo Clock Frequency Control {PMAC2 only}

Range 0 .. 15

Units Servo Clock Frequency = PHASE Clock Frequency / (I902+1)

Default 3 — SERVO Clock Frequency = 9.0346 kHz / (3+1) = 2.2587 kHz
(with default values of I900 and I901)

Remarks I902, in conjunction with I901 and I900, determines the frequency of the SERVO clock on PMAC2 (except for PMAC2 Ultralites, which use I992, I997, and I998 for this). Each cycle of the SERVO clock, PMAC2 updates the commanded position for each activated motor, and executes the servo algorithm to compute the command output to the amplifier.

Specifically, I902 controls how many times the SERVO clock frequency is divided down from the PHASE clock, whose frequency is set by I900 and I901. The SERVO clock frequency is equal to the PHASE clock frequency divided by (I902+1). I902 has a range of 0 to 15, so the frequency division can be by a factor of 1 to 16. The equation for I902 is:

$$I902 = \frac{PhaseFreq(kHz)}{ServoFreq(kHz)} - 1$$

The ratio of PHASE Clock Freq. to SERVO Clock Freq. must be an integer.

Note: If jumper E1 is ON, PMAC2 gets its SERVO clock signal externally from a serial-port input, and I902 is not used.

For execution of trajectories at the proper speed, I10 must be set properly to tell the trajectory generation software what the SERVO clock cycle time is. The formula for I10 is:

$$I10 = \frac{8{,}388{,}608}{ServoFreq(kHz)}$$

In terms of the variables that determine the SERVO clock frequency on a (non-Ultralite) PMAC2 board, the formula for I10 is:

$$I10 = \frac{640}{9}(2 * I900 + 3)(I901 + 1)(I902 + 1)$$

At the default servo clock frequency, I10 should be set to 3,713,707 in order that PMAC's interpolation routines use the proper servo update time.

> ***Note:***
>
> If the servo clock frequency is set too high, lower priority tasks such as communications can be starved for time. If the background tasks are completely starved, the watchdog timer will trip, shutting down the board. If a normal reset of the board does not re-establish a state where the watchdog timer has not tripped and communications works well, it will be necessary to re-initialize the board by powering up with the E3 re-initialization jumper on. This restores default settings, so communication is possible, and I900 and I901 can be set to supportable values.

Example With a 6.67 kHz PHASE Clock frequency established by I900 and I901, and a desired 3.33 kHz SERVO Clock frequency:
I902 = (6.67 / 3.33) - 1 = 2 - 1 = 1

See Also I10, I900, I901, I998

I903 Hardware Clock Control Channels 1-4 {PMAC2 only}

Range 0 .. 4095

Units I903 = Encoder SCLK Divider
..... + 8 * PFM_CLK Divider
..... + 64 * DAC_CLK Divider
..... + 512 * ADC_CLK Divider
.....where:
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ Encoder SCLK Divider)
PFM_CLK Frequency = 39.3216 MHz / (2 ^ PFM_CLK Divider)
DAC_CLK Frequency = 39.3216 MHz / (2 ^ DAC_CLK Divider)
ADC_CLK Frequency = 39.3216 MHz / (2 ^ ADC_CLK Divider)

Default 2258 = 2 + (8 * 2) + (64 * 3) + (512 * 4)
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
PFM_CLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
DAC_CLK Frequency = 39.3216 MHz / (2 ^ 3) = 4.9152 MHz
ADC_CLK Frequency = 39.3216 MHz / (2 ^ 4) = 2.4576 MHz

Remarks I903 controls the frequency of four hardware clock frequencies – SCLK, PFM_CLK, DAC_CLK, and ADC_CLK – for the first four machine interface channels on PMAC2. It is a 12-bit variable consisting of four independent 3-bit controls, one for each of the clocks. Each of these clock frequencies can be divided down from a starting 39.3216 MHz frequency by powers of 2, 2^N, from 1 to 128 times (N=0 to 7). This means that the possible frequency settings for each of these clocks are:

Frequency	Divide by	Divider N in $1/2^N$
39.3216 MHz	1	0
19.6608 MHz	2	1
9.8304 MHz	4	2
4.9152 MHz	8	3
2.4576 MHz	16	4
1.2288 MHz	32	5
614.4 kHz	64	6
307.2 kHz	128	7

Very few PMAC2 users will be required to change the setting of I903 from the default value.

The encoder sample clock signal SCLK controls how often PMAC2's digital hardware looks at the encoder and flag inputs. PMAC2 can take at most one count per SCLK cycle, so the SCLK frequency is the absolute maximum encoder count frequency. SCLK also controls the signal propagation through the digital delay filters for the encoders and flags; the lower the SCLK frequency, the greater the noise pulse that can be filtered out. The SCLK frequency should optimally be set to the lowest value that can accept encoder counts at the maximum possible rate.

Note:

If jumper E13 is ON in either setting, PMAC2 uses an external SCLK signal for encoder sampling and digital delay filter clocking; in this case, this part of I903 is not used.

The pulse-frequency-modulation clock PFM_CLK controls the PFM circuitry that is commonly used for stepper drives. The maximum pulse frequency possible is 1/4 of the PFM_CLK frequency. The PFM_CLK frequency should optimally be set to the lowest value that can generate pulses at the maximum frequency required.

The DAC_CLK controls the serial data frequency into D/A converters. If these converters are on Delta Tau-provided accessories, the DAC_CLK setting should be left at the default value.

The ADC_CLK controls the serial data frequency from A/D converters. If these converters are on Delta Tau-provided accessories, the ADC_CLK setting should be left at the default value.

To determine the clock frequencies set by a given value of I903, use the following procedure:

1. Divide I903 by 512 and round down to the nearest integer. This value N1 is the ADC_CLK divider.
2. Multiply N1 by 512 and subtract the product from I903 to get I903'. Divide I903' by 64 and round down to the nearest integer. This value N2 is the DAC_CLK divider.
3. Multiply N2 by 64 and subtract the product from I903' to get I903''. Divide I903'' by 8 and round down to the nearest integer. This value N3 is the PFM_CLK divider.

Multiply N3 by 8 and subtract the product from I903''. The resulting value N4 is the SCLK divider.

Example The maximum encoder count frequency in the application is 800 kHz, so the 1.2288 MHz SCLK frequency is chosen. A pulse train up to 500 kHz needs to be generated, so the 2.4576 MHz PFM_CLK frequency is chosen. The default serial DACs and ADCs provided by Delta Tau are used, so the default DAC_CLK frequency of 4.9152 MHz and the default ADC_CLK frequency of 2.4576 MHz are chosen. From the table:

.....SCLK Divider N: 5
.....PFM_CLK Divider N: 4
.....DAC_CLK Divider N: 3
.....ADC_CLK Divider N: 4
.....I903 = 5 + (8 * 4) + (64 * 3) + (512 * 4) = 5 + 32 + 192 + 2048 = 2277

I903 has been set to 3429. What clock frequencies does this set?

.....N1 = INT (3429/512) = 6	ADC_CLK = 611.44 kHz
.....I903' = 3429 - (512*6) = 357	
.....N2 = INT (357/64) = 5	DAC_CLK = 1.2288 MHz
.....I903'' = 357 - (64*5) = 37	
.....N3 = INT (37/8) = 4	PFM_CLK = 2.4576 MHz
.....N4 = 37 - (8*4) = 5	SCLK = 1.2288 MHz

See Also I907, I993

I904 PWM 1-4 Deadtime / PFM 1-4 Pulse Width Control {PMAC2 only}

Range 0 .. 255

Units PWM Deadtime = [16 / PWM_CLK (MHz)] * I904 = 0.135 usec * I904
PFM Pulse Width = [1 / PFM_CLK (MHz)] * I904
= PFM_CLK_period (usec) * I904

Default 15
PWM Deadtime = 0.135 usec * 15 = 2.03 usec
PFM Pulse Width = [1 / 9.8304 MHz] * 15 = 1.526 usec (with default I903)

Remarks I904 controls the deadtime period between top and bottom on-times in PMAC2's automatic PWM generation for machine interface channels 1-4. In conjunction with I903, it also controls the pulse width for PMAC2's automatic pulse-frequency modulation generation for machine interface channels 1-4.

The PWM deadtime, which is the delay between the top signal turning off and the bottom signal turning on, and vice versa, is specified in units of 16 PWM_CLK cycles. This means that the deadtime can be specified in increments of 0.135 usec. The equation for I904 as a function of PWM deadtime is:

$$I904 = \frac{DeadTime(\mu sec)}{0.135\mu sec}$$

The PFM pulse width is specified in PFM_CLK cycles, as defined by I903. The equation for I904 as a function of PFM pulse width and PFM_CLK frequency is:

$$I904 = PFM\_CLK\_Freq(MHz) * PFM\_Pulse\_Width(\mu sec)$$

In PFM pulse generation, the minimum off time between pulses is equal to the pulse width. This means that the maximum PFM output frequency is

$$PFM\_Max\_Freq(MHz) = \frac{PFM\_CLK\_Freq(MHz)}{2 * I904}$$

Example A PWM deadtime of approximately 1 microsecond is desired:

I904 ≅ 1 usec / 0.135 usec ≅ 7

With a 2.4576 MHz PFM_CLK frequency, a pulse width of 0.4 usec is desired:

I904 ≅ 2.4576 MHz * 0.4 usec ≅ 1

See Also I908, I994

I905 DAC 1-4 Strobe Word {PMAC2 only}

Range $000000 .. $FFFFFF

Units Serial Data Stream (MSB first, starting on rising edge of phase clock)

Default $7FFFC0

Remarks I905 controls the DAC strobe signal for machine interface channels 1-4. The 24-bit word set by I905 is shifted out serially on lines DAC_STROB1-4, MSB first, one bit per DAC_CLK cycle starting on the rising edge of the phase clock. The value in the LSB is held until the next phase clock cycle.

The default I905 value of $7FFFC0 is suitable for the 18-bit DACs on the ACC-8E Analog Interface Board. I905 should not be changed from the default unless different DACs are used.

For a 16-bit DAC, I905 should be set to $7FFF00. For a 12-bit DAC, I905 should be set to $7FF000.

See Also I909

I906 PWM 5-8 Frequency Control {PMAC2 only}

Range 0 .. 32767

Units PWM Frequency = 117,964.8 kHz / [4*I906+6]

Default 6257
PWM Frequency = 117,964.8 / 26114 = 4.5163 kHz

Remarks I906 controls the PWM frequency for machine interface channels 5-8. It does this by setting the limits of the PWM up-down counter, which increments and decrements at the PWMCLK frequency of 117,964.8 kHz (117.9648 MHz).

The PWM frequency determines the actual switching frequency of amplifiers connected to any of PMAC2's first four machine interface channels with the direct PWM command. The value of I906 is only important if the direct PWM command signal format is used on channels 5 to 8.

Generally, I906 is set to the same value as I900, which controls the frequency of channels 1 to 4. If a different PWM frequency is desired for channels 5 to 8, I906 should be set so that

$$\frac{2*PWM[5-8]Freq(kHz)}{PhaseFreq} = \{Integer\}$$

This will keep the PWM hardware on channels 5-8 in synchronization with the software algorithms driven by the PHASE clock, which is set by I900, I901, and I902. For example if the phase frequency is 10 kHz, the PWM frequency for channels 5 to 8 can be 5, 10, 15, 20, (etc.) kHz.

To set I906 for a desired PWM frequency, the following formula can be used:

$$I906 = \frac{117{,}964.8(kHz)}{4*PWM\_Freq(kHz)} - 1 \text{ (rounded down)}$$

Example A 30 kHz PWM frequency is desired for Channels 5-8:

$$I906 = (117{,}964.8 / [4 * 30]) - 1 = 982$$

See Also I900, I992

I907 Hardware Clock Control Channels 5-8 {PMAC2 only}

Range 0 .. 4095

Units I907 = Encoder SCLK Divider
.....+ 8 * PFM_CLK Divider
.....+ 64 * DAC_CLK Divider
.....+ 512 * ADC_CLK Divider
where:.....
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ Encoder SCLK Divider)
PFM_CLK Frequency = 39.3216 MHz / (2 ^ PFM_CLK Divider)
DAC_CLK Frequency = 39.3216 MHz / (2 ^ DAC_CLK Divider)
ADC_CLK Frequency = 39.3216 MHz / (2 ^ ADC_CLK Divider)

Default 2258 = 2 + (8 * 2) + (64 * 3) + (512 * 4)
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
PFM_CLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
DAC_CLK Frequency = 39.3216 MHz / (2 ^ 3) = 4.9152 MHz
ADC_CLK Frequency = 39.3216 MHz / (2 ^ 4) = 2.4576 MHz

Remarks I907 controls the frequency of four hardware clock frequencies for the second group of four machine interface channels on PMAC2 (channels 5-8). It is a 12-bit variable consisting of four independent 3-bit controls, one for each of the clocks. Each of these clock frequencies can be divided down from a starting 39.3216 MHz frequency by powers of 2, from 1 to 128 times.

This means that the possible frequency settings for each of these clocks are:

Frequency	Divide by	Divider N in 1/2^N
39.3216 MHz	1	0
19.6608 MHz	2	1
9.8304 MHz	4	2
4.9152 MHz	8	3
2.4576 MHz	16	4
1.2288 MHz	32	5
614.4 kHz	64	6
307.2 kHz	128	7

Very few PMAC2 users will be required to change the setting of I907 from the default value.

The encoder sample clock signal SCLK controls how often PMAC2's digital hardware looks at the encoder and flag inputs. PMAC2 can take at most one count per SCLK cycle, so the SCLK frequency is the absolute maximum encoder count frequency. SCLK also controls the signal propagation through the digital delay filters for the encoders and flags; the lower the SCLK frequency, the greater the noise pulse that can be filtered out. The SCLK frequency should optimally be set to the lowest value that can accept encoder counts at the maximum possible rate.

The pulse-frequency-modulation clock PFM_CLK controls the PFM circuitry that is commonly used for stepper drives. The maximum pulse frequency possible is 1/4 of the PFM_CLK frequency. The PFM_CLK frequency should optimally be set to the lowest value that can generate pulses at the maximum frequency required.

The DAC_CLK controls the serial data frequency into D/A converters. If these converters are on Delta Tau-provided accessories, the DAC_CLK setting should be left at the default value.

The ADC_CLK controls the serial data frequency from A/D converters. If these converters are on Delta Tau-provided accessories, the ADC_CLK setting should be left at the default value.

Example See I903 Example

See Also I903, I993

I908 PWM 5-8 Deadtime / PFM 5-8 Pulse Width Control {PMAC2 only}

Range 0 .. 255

Units PWM Deadtime = 0.135 usec * I908
PFM Pulse Width = [1 / PFM_CLK (MHz)] * I908
= PFM_CLK_period (usec) * I908

Default 15
PWM Deadtime = 0.135 usec * 15 = 2.03 usec
PFM Pulse Width = [1 / 9.8304 MHz] * 15 = 1.526 usec (with default I907)

Remarks I908 controls the deadtime period between top and bottom on-times in PMAC2's automatic PWM generation for machine interface channels 5-8. In conjunction with I907, it also controls the pulse width for PMAC2's automatic pulse-frequency modulation generation for machine interface channels 5-8.

The PWM deadtime, which is the delay between the top signal turning off and the bottom signal turning on, and vice versa, is specified in units of 16 PWM_CLK cycles. This means that the deadtime can be specified in increments of 0.135 usec. The equation for I908 as a function of PWM deadtime is:

$$I904 = \frac{DeadTime(\mu sec)}{0.135\mu sec}$$

The PFM pulse width is specified in PFM_CLK cycles, as defined by I907. The equation for I908 as a function of PFM pulse width and PFM_CLK frequency is:

$$I908 = PFM\_CLK\_Freq(MHz) * PFM\_Pulse\_Width(\mu sec)$$

In PFM pulse generation, the minimum off time between pulses is equal to the pulse width. This means that the maximum PFM output frequency is

$$PFM\_Max\_Freq(MHz) = \frac{PFM\_CLK\_Freq(MHz)}{2 * I904}$$

Example See I904 Example.

See Also I904, I994

I909 DAC 5-8 Strobe Word {PMAC2 only}

Range $000000 .. $FFFFFF

Units Serial Data Stream (MSB first, starting on rising edge of phase clock)

Default $7FFFC0

Remarks I909 controls the DAC strobe signal for machine interface channels 5-8. The 24-bit word set by I909 is shifted out serially on lines DAC_STROB1-4, MSB first, one bit per DAC_CLK cycle starting on the rising edge of the phase clock. The value in the LSB is held until the next phase clock cycle.

The default I909 value of $7FFFC0 is suitable for the 18-bit DACs on the ACC-8E Analog Interface Board. I909 should not be changed from the default unless different DACs are used.

For a 16-bit DAC, I909 should be set to $7FFF00. For a 12-bit DAC, I909 should be set to $7FF000.

See Also I905

Channel-Specific Gate Array I-Variables

(For Channel n, where n = 1 to 8)

I-Variables in the I910s through I980s control the hardware aspects of the "DSPGATE1" ASICs that provide the machine interface for channels 1 through 8. Each DSPGATE1 ASIC controls four channels. On an 8-channel PMAC2 (one that includes Option 1), I-variables for all 8 channels can be used. On a 4-channel PMAC2 (PMAC2-Lite or other PMAC2 without Option 1), only the I-variables for the first 4 channels can be used. On a PMAC2 Ultralite, there are no local machine interface channels, so none of the I-variables in this range may be used.

> ***Note:***
>
> In almost all cases, the machine interface channel n used for Motor x will be of the same number as the motor number (that is, n = x). However, this does not necessarily have to be the case, so it is a good idea to keep a clear distinction between the software motor functions and the hardware channel functions.

There are no I-variables for the Channels 9 – 16 that come on an ACC-24P/V2 board. Setup of these channels must be done with M-variables assigned to the appropriate control registers of these channels, and values assigned to these M-variables after every board power-up.

I9n0 Encoder/Timer n Decode Control {PMAC2 only}

Range 0 .. 15

Units None

Default 7

Remarks I9n0 controls how the input signal for Encoder n is decoded into counts. As such, this defines the sign and magnitude of a "count". The following settings may be used to decode an input signal.

Setting	Meaning
0	Pulse and direction CW
1	x1 quadrature decode CW
2	x2 quadrature decode CW
3	x4 quadrature decode CW
4	Pulse and direction CCW
5	x1 quadrature decode CCW
6	x2 quadrature decode CCW
7	x4 quadrature decode CCW
8	Internal pulse and direction
9	(reserved for future use)
10	(reserved for future use)
11	x6 hall decode CW
12	MLDT pulse timer control
13	(reserved for future use)
14	(reserved for future use)
15	x6 hall decode CCW

In any of the quadrature decode modes, PMAC2 is expecting two input waveforms on CHAn and CHBn, each with approximately 50% duty cycle, and approximately one-quarter of a cycle out of phase with each other. Times-one (x1) decode provides one count per cycle; x2 provides two counts per cycle; and x4 provides four counts per cycle. The vast majority of users select x4 decode to get maximum resolution.

The clockwise (CW) and counterclockwise (CCW) options simply control which direction counts up. If you get the wrong direction sense, simply change to the other option (e.g. from 7 to 3 or vice versa).

Note:

Changing the direction sense of the decode for the feedback encoder of a motor that is operating properly will result in unstable positive feedback and a dangerous runaway condition in the absence of other changes. The output polarity must be changed as well to re-establish polarity match for stable negative feedback.

In the pulse-and-direction decode modes, PMAC2 is expecting the pulse train on CHAn, and the direction (sign) signal on CHBn. If the signal is unidirectional, the CHBn line can be allowed to pull up to a high state, or it can be hardwired to a high or low state.

If I9n0 is set to 8, the decoder inputs the pulse and direction signal generated by Channel n's pulse frequency modulator (PFM) output circuitry. This permits the PMAC2 to create a phantom closed loop when driving an open-loop stepper system. *No jumpers or cables are needed to do this; the connection is entirely within the ASIC.* The counter polarity automatically matches the PFM output polarity.

If I9n0 is set to 11 or 15, the decoder looks at the 3-phase inputs on CHAn, CHBn, and CHCn, and decodes 6 states per cycle. This permits the use of hall-style commutation sensors for feedback. Each signal should be about 50% duty cycle, and 1/3-cycle offset from the other signals. The direction sense of the decode changes between I9n0 = 11 and I9n0 = 15. This mode is only supported on "B" and newer revisions of the DSPGATE1 IC.

If I9n0 is set to 12, the timer circuitry is set up to read magnetostrictive linear displacement transducers (MLDTs) such as Temposonics™. In this mode, the timer is cleared when the PFM circuitry sends out the excitation pulse to the sensor on PULSEn, and it is latched into the memory-mapped register when the excitation pulse is received on CHAn.

I9n1 Position Compare n Channel Select {PMAC2 only}

Range 0 .. 1

Units None

Default 0

Remarks I9n1 controls which encoder counter that Channel n's position compare circuitry operates with. When I9n1 is set to 0, the channel's position compare register is tied to the channel's own encoder counter, and the position compare signal appears only on the EQUn output.

When I9n1 is set to 1, the channel's position compare register is tied to the first encoder counter on the ASIC – Encoder 1 for channels 1-4, or Encoder 5 for channels 5-8 – and the position compare signal appears both on EQUn, and combined into the EQU output for the first channel on the IC (EQU1 or EQU5); executed as a logical OR.

I911 and I951 perform no effective function, so are always 1. They cannot be set to 0.

I9n2 Encoder n Capture Control {PMAC2 only}

Range 0 .. 15

Units none

Default 1

Remarks This parameter determines which input signal or combination of signals for channel n, and which polarity, triggers a hardware position capture of the counter for encoder n. If a flag input (home, limit, or user) is used, I9n3 determines which flag. Proper setup of this variable is essential for a successful home search, which depends on the position-capture function. The following settings may be used:

Setting	Meaning
0	Software Control (immediate capture)
1	Rising edge of CHCn (third channel)
2	Rising edge of Flag n (as set by Flag Select)
3	Rising edge of [CHCn AND Flag n] – Low true index, high true Flag
4	Software Control (immediate capture)
5	Falling edge of CHCn (third channel)
6	Rising edge of Flag n (as set by Flag Select)
7	Rising edge of [CHCn/ AND Flag n] – Low true index, high true Flag
8	Software Control (immediate capture)
9	Rising edge of CHCn (third channel)
10	Falling edge of Flag n (as set by Flag Select)
11	Rising edge of [CHCn AND Flag n/] – High true index, low true Flag
12	Software Control (immediate capture)
13	Falling edge of CHCn (third channel)
14	Falling edge of Flag n (as set by Flag Select)
15	Rising edge of [CHCn/ AND Flag n/] – Low true index, low true Flag

Note that only flags and index inputs of the same channel number as the encoder may be used for hardware capture of that encoder's position. This means that to use the hardware capture feature for the homing search move, Ix25 must use flags of the same channel number as the encoder that Ix03 uses for position-loop feedback.

To do a software-controlled position capture, preset this parameter to 0 or 8; when the parameter is then changed to 4 or 12, the capture is triggered (this is not of much practical use).

The trigger is armed when the position capture register is read. After this, as soon as PMAC2 sees that the specified input lines are in the specified states, the trigger will occur – it is level-triggered, not edge-triggered.

I9n3 Capture n Flag Select Control {PMAC2 only}

Range 0 .. 3

Units none

Default 0

Remarks This parameter determines which of the "Flag" inputs will be used for position capture (if one is used – see I902 etc.):

.....0: HMFLn (Home Flag n)
.....1: PLIMn (Positive End Limit Flag n)
.....2: MLIMn (Negative End Limit Flag n)
.....3: USERn (User Flag n)

Typically, this parameter is set to zero, because in actual use the LIMn flags create other effects that usually interfere with what is trying to be accomplished by the position capture. If you wish to capture on the PLIMn or MLIMn flags, you probably will want to disable their normal functions with Ix25, or use a channel n where none of the flags is used for the normal axis functions.

I9n4 Encoder n Gated Index Select {PMAC2 only}

Range 0 .. 1

Units none

Default 0

Remarks I9n4 controls whether the "raw" encoder index signal is used for the position capture of the channels' encoder counter, or whether the quadrature signals of the encoder are first used to create a pulse that is a single quadrature state wide. When I9n4 is set to 0, the encoder index channel input (CHCn) is passed directly into the position capture circuitry.

When I9n4 is set to 1, the encoder index channel input (CHCn) is logically combined with ("gated by") the quadrature signals of Encoder n before going to the position capture circuitry. The intent is to get a "gated index" signal exactly one quadrature state wide. This provides a more accurate and repeatable capture, and makes the use of the capture function to confirm the proper number of counts per revolution very straightforward.

In order for the gated index capture to work reliably, the index pulse must reliably span one, but only one, "high-high" or "low-low" AB quadrature state of the encoder. I9n5 allows you to select which of these two possibilities is used.

Note:

If I9n4 is set to 1, but I9n2 bit 0 is set to 0, so the index is not used in the position capture, then the encoder position is captured on the first edge of any of the U, V, or W flag inputs for the channel. In this case, bits 0, 1, and 2 of the channel status word tell what hall-state edge caused the capture.

I9n5 Channel n Encoder Index Gate State/Demux Control {PMAC2 only}

Range 0 - 3

Units none

Default 0

Remarks I9n5 is a 2-bit variable that controls two functions for the index channel of the encoder.

When using the "gated index" feature of a PMAC2 Servo IC for more accurate position capture (I9n4=1), bit 0 of I9n5 specifies whether the raw index-channel signal fed into Encoder n is passed through to the position capture signal only on the "high-high" quadrature state (bit 0 = 0), or only on the "low-low" quadrature state (bit 0 = 1).

Bit 1 of I9n5 controls whether the Servo IC "de-multiplexes" the index pulse and the 3 hall-style commutation states from the third channel based on the quadrature state, as with Yaskawa incremental encoders. If bit 1 is set to 0, this de-multiplexing function is not performed, and the signal on the "C" channel of the encoder is used as the index only. If bit 1 is set to 1, the Servo IC breaks out the third-channel signal into four separate values, one for each of the four possible AB-quadrature states. The de-multiplexed hall commutation states can be used to provide power-on phase position using Ix81.

Note: The "B" revision or newer of the DSPGATE1 Servo IC is required to support this hall de-multiplexing feature.

Note: Immediately after power-up, the Yaskawa encoder automatically cycles its AB outputs forward and back through a full quadrature cycle to ensure that all of the hall commutation states are available to the controller before any movement is started. However, if the encoder is powered up at the same time as the PMAC2, this will happen before the Servo IC is ready to accept these signals. Bit 2 of the channel's status word, "Invalid De-multiplex", will be set to 1 if the Servo IC has not seen all of these states when it was ready for them. To use this feature, it is recommended that the power to the encoder be provided through a software-controlled relay to ensure that valid readings of all states have been read before using these signals for power-on phasing.

I9n5 has the following possible settings:

- I9n5 = 0: Gate index with "high-high" quadrature state (GI = A & B & C), no demux
- I9n5 = 1: Gate index with "low-low" quadrature state (GI = A/ & B/ & C), no demux
- I9n5 = 2 or 3: De-multiplex hall and index from third channel, gating irrelevant

Note: Prior to firmware revision V1.17C, I9n5 was a single-bit I-variable controlling the gating state only. The control bit for the de-multiplexing function had to be accessed directly with an M-variable (it was stored to flash on a **SAVE** command and restored on power-up/reset).

I9n6 Output n Mode Select {PMAC2 only}

Range 0 .. 3

Units none

Default 0

Remarks 0 = Outputs A & B are PWM; Output C is PWM
1 = Outputs A & B are DAC; Output C is PWM
2 = Outputs A & B are PWM; Output C is PFM
3 = Outputs A & B are DAC; Output C is PFM

I9n6 controls what output formats are used on the command output signal lines for

machine interface channel n. If a three-phase direct PWM command format is desired, I9n6 should be set to 0. If signal outputs for (external) digital-to-analog converters are desired, I9n6 should be set to 1 or 3. In this case, the C output can be used as a supplemental (non-servo) output in either PWM or PFM form. For example, it can be used to excite an MLDT sensor (e.g. Temposonics™) in PFM form.

I9n7 Output n Invert Control {PMAC2 only}

Range 0 .. 3

Units none

Default 0

Remarks I9n7 controls the high/low polarity of the command output signals for Channel n. The default non-inverted outputs are high true.

For PWM signals on Outputs A, B, and C, this means that the transistor-on signal is high. Delta Tau PWM-input amplifiers, and most other PWM-input amplifiers, expect this non-inverted output format. For such a 3-phase motor drive, I9n7 should be set to 0.

Note:

If the high/low polarity of the PWM signals is wrong for a particular amplifier, what was intended to be deadtime between top and bottom on-states as set by I904 and I908 becomes overlap. If the amplifier input circuitry does not lock this out properly, this causes an effective momentary short circuit between bus power and ground. This would destroy the power transistors very quickly.

For PFM signals on Output C, non-inverted means that the pulse-on signal is high (direction polarity is controlled by I9n8). During a change of direction, the direction bit will change synchronously with the leading edge of the pulse, which in the non-inverted form is the rising edge.

If the drive requires a set-up time on the direction line before the rising edge of the pulse, the pulse output can be inverted so that the rising edge is the trailing edge, and the pulse width (established by I904 or I908) is the set-up time.

For DAC signals on Outputs A and B, non-inverted means that a 1 value to the DAC is high. DACs used on Delta Tau accessory boards, as well as all other known DACs always expect non-inverted inputs, so I9n7 should always be set to 0 or 2 when using DACs on Channel n.

Note:

Changing the high/low polarity of the digital data to the DACs has the effect of inverting the voltage sense of the DACs' analog outputs. This changes the polarity match between output and feedback. If the feedback loop had been stable with negative feedback, this change would create destabilizing positive feedback, resulting in a dangerous runaway condition that would only be stopped when the motor exceeded Ix11 fatal following error.

I9n8 Output n PFM Direction Signal Invert Control {PMAC2 only}

Range 0 .. 1

Units none

Default 0

Remarks 0 = Do not invert direction signal (+ = low; - = high)

1 = Invert direction signal (- = low; + = high)

I9n8 controls the polarity of the direction output signal in the pulse-and-direction format for Channel n. It is active only if I9n6 has been set to 2 or 3 to use Output C as a pulse-frequency-modulated (PFM) output.

If I9n8 is set to the default value of 0, a positive direction command provides a low output; if I9n8 is set to 1, a positive direction command provides a high output.

I9n9 Channel n Hardware-1/T Control {PMAC2 only}

Range 0 – 1

Units none

Default 0

Remarks I9n9 controls whether the "hardware-1/T" functionality is enabled for a PMAC2 Servo IC on Channel n. If I9n9 is set to the default value of 0, the hardware-1/T functionality is disabled, permitting the use of the "software-1/T" position extension that is calculated by default with encoder conversion method $0. If I9n9 is set to 1, the hardware-1/T functionality is enabled (if present on the IC), and the software-1/T cannot be used.

The hardware-1/T functionality is present only on Revision D and newer of the PMAC2-style DSPGATE1 IC, released at the beginning of the year 2002. Setting I9n9 to 1 on an older revision IC does nothing – software-1/T functions can still be used.

When the hardware-1/T functionality is enabled, the IC computes a new fractional-count position estimate based on timers every SCLK (encoder sample clock) cycle. This permits the fractional count data to be used for hardware capture and compare functions, enhancing their resolution. This is particularly useful when the IC is used on an ACC-51 high-resolution analog-encoder interpolator board. However, it replaces the timer registers at the first two "Y" addresses for the channel with fractional count position data, so the traditional software-1/T method of the conversion table cannot work if this is enabled.

If you enable the hardware-1/T functionality, and want to be able to use 1/T interpolation in your servo loop, you must use the hardware-1/T extension method ($C method digit with the mode switch bit set to 1) in the encoder conversion table.

PMAC2 DSPGATE2 I-Variables

I-Variables numbered in the I990s control hardware aspects of the "DSPGATE2" ASIC. This IC controls operation of the MACRO ring on all PMAC2 boards. On the Ultralite versions of the PMAC2, this IC also controls the frequency of the clock signals on the board, because the "DSPGATE1" ICs are not present. On all of these boards, I990 and I991 control the decode of the handwheel encoder inputs on the JHW port.

I990 Handwheel 1 Decode Control {PMAC2 only}

Range 0 .. 15

Units none

Default 7

Remarks I990 controls how the input signal for Handwheel 1 on the JHW port is decoded into counts. As such, this defines the sign and magnitude of a "count". The following settings may be used to decode an input signal.

0: .....Pulse and direction CW
1: .....x1 quadrature decode CW
2: .....x2 quadrature decode CW

3: .....x4 quadrature decode CW
4: .....Pulse and direction CCW
5: .....x1 quadrature decode CCW
6: .....x2 quadrature decode CCW
7: .....x4 quadrature decode CCW
8: .....Internal pulse and direction
9-11: .....Not used
12: .....MLDT pulse timer control
13-15: .....Not used

In any of the quadrature decode modes, PMAC2 is expecting two input waveforms on HWA1 and HWB1, each with approximately 50% duty cycle, and approximately one-quarter of a cycle out of phase with each other. "Times-one" (x1) decode provides one count per cycle; x2 provides two counts per cycle; and x4 provides four counts per cycle. The vast majority of users select x4 decode to get maximum resolution.

The "clockwise" (CW) and "counterclockwise" (CCW) options simply control which direction counts up. If you get the wrong direction sense, simply change to the other option (e.g. from 7 to 3 or vice versa)

Note:

Changing the direction sense of the decode for the feedback encoder of a motor that is operating properly will result in unstable positive feedback and a dangerous runaway condition in the absence of other changes. The output polarity must be changed as well to re-establish polarity match for stable negative feedback.

In the pulse-and-direction decode modes, PMAC2 is expecting the pulse train on HWA1, and the direction (sign) signal on HWB1. If the signal is unidirectional, the HWB1 line can be allowed to pull up to a high state, or it can be hardwired to a high or low state.

If I990 is set to 8, the decoder inputs the pulse and direction signal generated by Channel 1*'s pulse frequency modulator (PFM) output circuitry. This permits the PMAC2 to create a phantom closed loop when driving an open-loop stepper system. No jumpers or cables are needed to do this; the connection is entirely within the ASIC. The counter polarity automatically matches the PFM output polarity. This mode is only supported on "B" and newer revisions of the DSPGATE2 IC.

If I990 is set to 12, the timer circuitry is set up to read magnetostrictive linear displacement transducers (MLDTs) such as Temposonics™. In this mode, the timer is cleared when the PFM circuitry sends out the excitation pulse to the sensor on PULSEn, and it is latched into the memory-mapped register when the excitation pulse is received on HWA1. This mode is only supported on "B" and newer revisions of the DSPGATE2 IC.

See Also I9n0, I991

I991 Handwheel 2 Decode Control {PMAC2 only}

Range 0 .. 15

Units none

Default 7

Remarks I991 controls how the input signal for Handwheel 2 is decoded into counts. As such, this defines the sign and magnitude of a "count". The following settings may be used to decode an input signal.

0: .....Pulse and direction CW

1: .....x1 quadrature decode CW
2: .....x2 quadrature decode CW
3: .....x4 quadrature decode CW
4: .....Pulse and direction CCW
5: .....x1 quadrature decode CCW
6: .....x2 quadrature decode CCW
7: .....x4 quadrature decode CCW
8: .....Internal pulse and direction
9-11: .....Not used
12: .....MLDT pulse timer control
13-15: .....Not used

In any of the quadrature decode modes, PMAC2 is expecting two input waveforms on HWA2 and HWB2, each with approximately 50% duty cycle, and approximately one-quarter of a cycle out of phase with each other. "Times-one" (x1) decode provides one count per cycle; x2 provides two counts per cycle; and x4 provides four counts per cycle. The vast majority of users select x4 decode to get maximum resolution.

The "clockwise" (CW) and "counterclockwise" (CCW) options simply control which direction counts up. If you get the wrong direction sense, simply change to the other option (e.g. from 7 to 3 or vice versa)

Note:

Changing the direction sense of the decode for the feedback encoder of a motor that is operating properly will result in unstable positive feedback and a dangerous runaway condition in the absence of other changes. The output polarity must be changed as well to re-establish polarity match for stable negative feedback.

In the pulse-and-direction decode modes, PMAC2 is expecting the pulse train on HWA2, and the direction (sign) signal on HWB2. If the signal is unidirectional, the HWB2 line can be allowed to pull up to a high state, or it can be hardwired to a high or low state.

If I991 is set to 8, the decoder inputs the pulse and direction signal generated by Channel 2*'s pulse frequency modulator (PFM) output circuitry. This permits the PMAC2 to create a phantom closed loop when driving an open-loop stepper system. *No jumpers or cables are needed to do this; the connection is entirely within the ASIC.* The counter polarity automatically matches the PFM output polarity. This mode is only supported on "B" and newer revisions of the DSPGATE2 IC.

If I991 is set to 12, the timer circuitry is set up to read magnetostrictive linear displacement transducers (MLDTs) such as Temposonics™. In this mode, the timer is cleared when the PFM circuitry sends out the excitation pulse to the sensor on PULSEn, and it is latched into the memory-mapped register when the excitation pulse is received on HWA2. This mode is only supported on "B" and newer revisions of the DSPGATE2 IC.

See Also I9n0, I990

I992 MaxPhase and PWM 1*-2* Frequency Control {PMAC2 only}

Range 0 .. 32767

Units MaxPhase Frequency = 117,964.8 kHz / [2*I992+3]
PWM Frequency = 117,964.8 kHz / [4*I992+6]

Default 6527
MaxPhase Frequency = 117,964.8 / 13057 = 9.0346 kHz

PWM Frequency = 117,964.8 / 26114 = 4.5173 kHz

> ***Note:***
>
> On PMAC2 boards that are not "Ultralite", I992 does not control the MaxPhase frequency; I900 does. On all PMAC2 boards, the PWM 1*-2* frequency is only important if you are using supplemental PWM channels.

Remarks I992 controls the maximum phase clock frequency for the PMAC2 Ultralite, and the PWM frequency for supplementary machine interface channels 1* and 2*. It does this by setting the limits of the PWM up-down counter, which increments and decrements at the PWMCLK frequency of 117,964.8 kHz (117.9648 MHz).

The actual phase clock frequency is divided down from the maximum phase clock according to the setting of I997. On the falling edge of the phase clock, PMAC2 Ultralite starts transmission of a set of MACRO ring data and interrupts the processor to start any necessary phase commutation and digital current-loop algorithms. Even if phasing and current-loop algorithms are not used, the MaxPhase and Phase clock frequencies are important because the servo clock is derived from the phase clock.

To set I992 for a desired "maximum phase" clock frequency, the following formula can be used:

*I992 = (117,964.8 kHz / [2*MaxPhase (kHz)]) - 1 (rounded down)*

On PMAC2 boards that are not "Ultralite", I992 is generally set to the same value as I900, which controls the maximum phase frequency, and the PWM frequency of channels 1 to 4. If a different PWM frequency is desired for channels 1* and 2*, I992 should be set so that

$$\frac{2 * PWM[1*-2*]Freq(kHz)}{PhaseFreq} = \{Integer\}$$

Example To set a PWM frequency of 10 kHz and therefore a MaxPhase clock frequency of 20 kHz:

I992 = (117,964.8 kHz / [4*10 kHz]) - 1 = 2948

To set a PWM frequency of 7.5 kHz and therefore a MaxPhase clock frequency of 15 kHz:
I992 = (117,964.8 kHz / [4*7.5 kHz]) - 1 = 3931

I993 Hardware Clock Control Channels 1*-2* {PMAC2 only}

Range 0 .. 4095

Units I993 = Encoder SCLK Divider
.....+ 8 * PFM_CLK Divider
.....+ 64 * DAC_CLK Divider
.....+ 512 * ADC_CLK Divider
.....where:
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ Encoder SCLK Divider)
PFM_CLK Frequency = 39.3216 MHz / (2 ^ PFM_CLK Divider)
DAC_CLK Frequency = 39.3216 MHz / (2 ^ DAC_CLK Divider)
ADC_CLK Frequency = 39.3216 MHz / (2 ^ ADC_CLK Divider)

Default 2258 = 2 + (8 * 2) + (64 * 3) + (512 * 4)
Encoder SCLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
PFM_CLK Frequency = 39.3216 MHz / (2 ^ 2) = 9.8304 MHz
DAC_CLK Frequency = 39.3216 MHz / (2 ^ 3) = 4.9152 MHz

ADC_CLK Frequency = 39.3216 MHz / (2 ^ 4) = 2.4576 MHz

Remarks I993 controls the frequency of three hardware clock frequencies – SCLK, PFM_CLK, and ADC_CLK – for the supplemental machine interface channels 1* and 2* on PMAC2 or PMAC2 Ultralite (there is no DAC_CLK on the supplemental channels, but it is referred to here for consistency with I903 and I907). It is a 12-bit variable consisting of four independent 3-bit controls (the 3 bits for DAC_CLK are "don't care"), one for each of the clocks. Each of these clock frequencies can be divided down from a starting 39.3216 MHz frequency by powers of 2, 2^N, from 1 to 128 times (N=0 to 7). This means that the possible frequency settings for each of these clocks are:

Frequency	Divide by	Divider N in $1/2^N$
39.3216 MHz	1	0
19.6608 MHz	2	1
9.8304 MHz	4	2
4.9152 MHz	8	3
2.4576 MHz	16	4
1.2288 MHz	32	5
614.4 kHz	64	6
307.2 kHz	128	7

Very few PMAC2 users will be required to change the setting of I993 from the default value.

The encoder sample clock signal SCLK controls how often PMAC2's digital hardware looks at the handwheel encoder inputs. PMAC2 can take at most one count per SCLK cycle, so the SCLK frequency is the absolute maximum encoder count frequency. SCLK also controls the signal propagation through the digital delay filters for the encoders and flags; the lower the SCLK frequency, the greater the noise pulse that can be filtered out. The SCLK frequency should optimally be set to the lowest value that can accept encoder counts at the maximum possible rate.

The pulse-frequency-modulation clock PFM_CLK controls the PFM circuitry that can create pulse and direction outputs on the JHW connector. The maximum pulse frequency possible is 1/4 of the PFM_CLK frequency. The PFM_CLK frequency should optimally be set to the lowest value that can generate pulses at the maximum frequency required.

The ADC_CLK controls the serial data frequency from A/D converters. These can only be accessed as the alternate use of general-purpose I/O pins.

To determine the clock frequencies set by a given value of I993, use the following procedure:

1. Divide I993 by 512 and round down to the nearest integer. This value N1 is the ADC_CLK divider.
2. Multiply N1 by 512 and subtract the product from I993 to get I993'. Divide I993' by 64 and round down to the nearest integer. This value N2 is the DAC_CLK divider (not relevant here).
3. Multiply N2 by 64 and subtract the product from I993' to get I993''. Divide I993'' by 8 and round down to the nearest integer. This value N3 is the PFM_CLK divider.
4. Multiply N3 by 8 and subtract the product from I993''. The resulting value N4 is the

SCLK divider.

Example The maximum encoder count frequency in the application is 800 kHz, so the 1.2288 MHz SCLK frequency is chosen. A pulse train up to 500 kHz needs to be generated, so the 2.4576 MHz PFM_CLK frequency is chosen. ADCs and DACs are not used, so the default DAC_CLK frequency of 4.9152 MHz and the default ADC_CLK frequency of 2.4576 MHz are chosen. From the table:

.....SCLK Divider N: 5
.....PFM_CLK Divider N: 4
.....DAC_CLK Divider N: 3
.....ADC_CLK Divider N: 4
.....I993 = 5 + (8 * 4) + (64 * 3) + (512 * 4) = 5 + 32 + 192 + 2048 = 2277

I993 has been set to 3429. What clock frequencies does this set?

.....N1 = INT (3429/512) = 6 ADC_CLK = 611.44 kHz
.....I993' = 3429 - (512*6) = 357
.....N2 = INT (357/64) = 5 DAC_CLK = 1.2288 MHz
.....I993'' = 357 - (64*5) = 37
.....N3 = INT (37/8) = 4 PFM_CLK = 2.4576 MHz
N4 = 37 - (8*4) = 5 SCLK = 1.2288 MHz

I994 PWM 1*-2* Deadtime / PFM 1* Pulse Width Control {PMAC2 only}

Range 0 .. 255

Units PWM Deadtime = [16 / PWM_CLK (MHz)] * I994 = 0.135 usec * I994
PFM Pulse Width = [1 / PFM_CLK (MHz)] * I994
= PFM_CLK_period (usec) * I994

Default 15
PWM Deadtime = 0.135 usec * 15 = 2.03 usec
PFM Pulse Width = [1 / 9.8304 MHz] * 15 = 1.526 usec (with default I993)

Remarks I994 controls the deadtime period between top and bottom on-times in PMAC2's automatic PWM generation for supplemental machine interface channels 1* and 2*. In conjunction with I993, it also controls the pulse width for PMAC2's automatic pulse-frequency modulation generation for supplemental machine interface channel 1*.

The PWM deadtime, which is the delay between the top signal turning off and the bottom signal turning on, and vice versa, is specified in units of 16 PWM_CLK cycles. This means that the deadtime can be specified in increments of 0.135 usec. The equation for I994 as a function of PWM deadtime is:

$$I994 = \frac{DeadTime(\mu sec)}{0.135\mu sec}$$

The PFM pulse width is specified in PFM_CLK cycles, as defined by I993. The equation for I994 as a function of PFM pulse width and PFM_CLK frequency is:

$$I994 = PFM\_CLK\_Freq(MHz) * PFM\_Pulse\_Width(\mu sec)$$

In PFM pulse generation, the minimum off time between pulses is equal to the pulse width. This means that the maximum PFM output frequency is:

$$PFM\_Max\_Freq(MHz) = \frac{PFM\_CLK\_Freq(MHz)}{2 * I994}$$

Example A PWM deadtime of approximately 1 microsecond is desired:

I994 ≅ 1 usec / 0.135 usec ≅ 7

With a 2.4576 MHz PFM_CLK frequency, a pulse width of 0.4 usec is desired:

I994 ≅ 2.4576 MHz * 0.4 usec ≅ 1

I995 MACRO Ring Configuration/Status {PMAC2 only}

Range $0000 .. $FFFF (0 - 65,535)

Units none

Default 0

Remarks I995 contains configuration and status bits for MACRO ring operation of the PMAC2. There are 11 configuration bits and 5 status bits, as follows:

Bit #	Value	Type	Function
0	1($1)	Status	Data Overrun Error (cleared when read)
1	2($2)	Status	Byte Violation Error (cleared when read)
2	4($4)	Status	Packet Parity Error (cleared when read)
3	8($8)	Status	Packet Underrun Error (cleared when read)
4	16($10)	Config	Master Station Enable
5	32($20)	Config	Synchronizing Master Station Enable
6	64($40)	Status	Sync Node Packet Received (cleared when read)
7	128($80)	Config	Sync Node Phase Lock Enable
8	256($100)	Config	Node 8 Master Address Check Disable
9	512($200)	Config	Node 9 Master Address Check Disable
10	1024($400)	Config	Node 10 Master Address Check Disable
11	2048($800)	Config	Node 11 Master Address Check Disable
12	4096($1000)	Config	Node 12 Master Address Check Disable
13	8192($2000)	Config	Node 13 Master Address Check Disable
14	16384($4000)	Config	Node 14 Master Address Check Disable
15	32768($8000)	Config	Node 15 Master Address Check Disable

In most applications, the only important configuration bits are bits 4, 5, and 7. In every MACRO ring, there must be one and only one synchronizing master. On this card, bits 4 and 5 should be set (1), but bit 7 should be clear (0). On this card, I995 should be set to $30, or $xx30 if any of the high bits are to be set.

If there are more that one PMAC2 acting as a masters on the ring, these should not be synchronizing masters, but they should enable “sync node phase lock” to stay synchronized with the synchronizing master. One these cards, bit 4 should be set, bit 5 should be clear, and bit 7 should be set, so I995 should be set to $90, or $xx90 if any of the high bits are to be set.

Bits 8-15 can be set individually to disable the “master address check” for their corresponding node numbers. This capability is for multi-master broadcast and synchronization. If the master address check is disabled, only the slave node number part of the packet address must match for a packet to be latched in. In this way, the synchronizing master can send the same data packet to multiple other master and slave stations. This common packet can be used to keep multiple stations synchronized using the sync lock function enabled with bit 7 of I995; the packet number is specified in I996 (packet 15 is suggested for this purpose).

I996 MACRO Node Activate Control {PMAC2 only}

Range $000000 .. $FFFFFF (0 to 8,388,607)

Units none

Default $0 (all nodes de-activated)

Remarks I996 controls which of the 16 MACRO nodes on the card are activated. It also controls the master station number, and the node number of the packet that creates a synchronization signal. The bits of I996 are arranged as follows:

Bit #	Value	Type	Function
0	1($1)	Config	Node 0 Activate
1	2($2)	Config	Node 1 Activate
2	4($4)	Config	Node 2 Activate
3	8($8)	Config	Node 3 Activate
4	16($10)	Config	Node 4 Activate
5	32($20)	Config	Node 5 Activate
6	64($40)	Config	Node 6 Activate
7	128($80)	Config	Node 7 Activate
8	256($100)	Config	Node 8 Activate
9	512($200)	Config	Node 9 Activate
10	1024($400)	Config	Node 10 Activate
11	2048($800)	Config	Node 11 Activate
12	4096($1000)	Config	Node 12 Activate
13	8192($2000)	Config	Node 13 Activate
14	16384($4000)	Config	Node 14 Activate
15	32768($8000)	Config	Node 15 Activate
16-19	$X0000	Config	Packet Sync Node Slave Address (0 - 15)
20-23	$X00000	Config	Master Station Number (0-15)

Bits 0 to 15 are individual control bits for the matching node number 0 to 15. If the bit is set to 1, the node is activated; if the bit is set to 0, the node is de-activated.

Note:

If the use of an activated node n includes auxiliary register functions, including servo flags, bit n of I1000 must also be set to 1.

If the PMAC2 is a master station (likely) as determined by I995, it will send out a packet for each activated node every ring cycle (every phase cycle). When it receives a packet for an activated node, it will latch in that packet and not pass anything on.

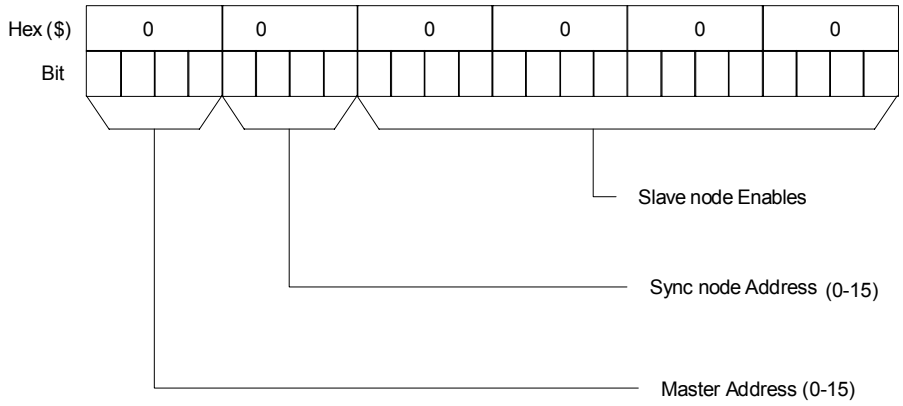
If the PMAC2 is a slave station (unlikely but possible) as determined by I995, when it receives a packet for an activated node, it will latch in the contents of that packet into its read registers for that node address, and automatically substitute the contents of its write registers into the packet.

If a node is disabled, the PMAC2, whether master or slave, will still latch in the contents of a packet it receives, but it will also pass on the packet unchanged. This feature is particularly useful for the MACRO broadcast feature, in which multiple stations need to receive the same packet.

Bits 16-19 together specify the slave number part of the packet address (0-15) that will cause a sync lock pulse on the card, if this function is enabled by I995. This function is useful for a PMAC2 that is a slave or non-synchronizing master on the ring, to keep it locked to the synchronizing master. If the master address check for this node is disabled with I995, only the slave number must match to create the sync lock pulse. If the master address check is left enabled, the master number part of the packet address must match the master number for the card, as set in bits 20-23 of I996.

If this card is the synchronizing master, this function is not enabled, so the value of these bits does not matter; they can be left at the default of 0.

Bits 20-23 specify the master number for the card (0-15). The number must be specified whether the card is a master station or a slave station.



Note:

On prototype PMAC2 boards that did not support multi-master MACRO rings, I996 contained only bits 0-15.

Example Master number 0; Sync node address 0
Activated nodes 0-5; De-activated nodes 6-15:
I996 =0000 0000 0000 0000 0011 1111 (binary) = $00003F
Master number 1; Sync node address 15 ($F)
Activated nodes 0, 2, 4, 6, 8, 10, 12; other nodes de-activated:
I996 = 0001 1111 0001 0101 0101 0101 (binary) = $1F1555

I997 Phase Clock Frequency Control {PMAC2 only}

Range 0 .. 15

Units PHASE Clock Frequency = MaxPhase Frequency / (I997+1)

Default 0
PHASE Clock Frequency = 9.0346 kHz / 1 = 9.0346 kHz
(with default value of I992)

Remarks I997, in conjunction with I992, determines the frequency of the PHASE clock on PMAC2 Ultralite. Each cycle of the PHASE clock, a set of MACRO ring information is transmitted, and any required motor phase commutation and digital current-loop algorithms are performed for specified motors.

Note:

On PMAC2 boards that are not "Ultralite", I997 does not control the Phase Clock frequency; I901 does. I997 has no effect on non-Ultralite versions of the PMAC2.

Specifically, I997 controls how many times the PHASE clock frequency is divided down from the maximum phase clock, whose frequency is set by I992. The PHASE clock frequency is equal to the maximum phase clock frequency divided by (I997+1). I997 has a range of 0 to 15, so the frequency division can be by a factor of 1 to 16. The equation for I997 is:

$$I997 = \frac{MaxPhaseFreq(kHz)}{PhaseFreq(kHz)} - 1$$

The ratio of MaxPhase Freq. to PHASE Clock Freq. must be an integer.

Note:

If jumper E1 is ON, PMAC2 Ultralite gets its PHASE clock signal externally from a serial-port input, and I997 is not used.

Note:

If the phase clock frequency is set too high, lower priority tasks such as communications can be starved for time. If the background tasks are completely starved, the watchdog timer will trip, shutting down the board. If a normal reset of the board does not re-establish a state where the watchdog timer has not tripped and communications works well, it will be necessary to re-initialize the board by powering up with the E3 re-initialization jumper on. This restores default settings, so communication is possible, and I992 and I997 can be set to supportable values.

Example With a 20 kHz MaxPhase Clock frequency established by I992, and a desired 6.67 kHz PHASE clock frequency, the ratio between MaxPhase and PHASE is 3:

I997 = (20 / 6.67) - 1 = 3 -1 = 2

I998 Servo Clock Frequency Control {PMAC2 only}

Range 0 .. 15

Units Servo Clock Frequency = PHASE Clock Frequency / (I998+1)

Default 3
SERVOClock Frequency = 9.0346 kHz / (3+1) = 2.2587 kHz
(with default values of I992 and I997)

Remarks I998, in conjunction with I997 and I992, determines the frequency of the SERVO clock on PMAC2 Ultralite. Each cycle of the SERVO clock, PMAC2 Ultralite updates the commanded position for each activated motor, and executes the servo algorithm to compute the command output to the amplifier.

Note:

On PMAC2 boards that are not "Ultralite", I998 does not control the Servo Clock frequency; I902 does. I998 has no effect on non-Ultralite versions of the PMAC2.

Specifically, I998 controls how many times the SERVO clock frequency is divided down from the PHASE clock, whose frequency is set by I992 and I997. The SERVO clock frequency is equal to the PHASE clock frequency divided by (I998+1). I998 has a range of 0 to 15, so the frequency division can be by a factor of 1 to 16. The equation for I998 is:

$$I998 = \frac{PhaseFreq(kHz)}{ServoFreq(kHz)} - 1$$

The ratio of PHASE Clock Freq. to SERVO Clock Freq. must be an integer.

Note:

If jumper E1 is ON, PMAC2 Ultralite gets its SERVO clock signal externally from a serial-port input, and I998 is not used.

For execution of trajectories at the proper speed, I10 must be set properly to tell the trajectory generation software what the SERVO clock cycle time is. The formula for I10 is:

$$I10 = \frac{8{,}388{,}608}{ServoFreq(kHz)}$$

In terms of the variables that determine the SERVO clock frequency on a PMAC2 Ultralite board, the formula for I10 is:

$$I10 = \frac{640}{9}(2 * I992 + 3)(I997 + 1)(I998 + 1)$$

Example With a 6.67 kHz PHASE Clock frequency established by I900 and I997, and a desired 3.33 kHz SERVO Clock frequency:

$$I998 = (6.67 / 3.33) - 1 = 2 - 1 = 1$$

I999 (Reserved for Future Use)

MACRO Software Setup I-Variables

I1000 MACRO Node Auxiliary Register Enable

Range 0 .. $FFFF (0 .. 65,535)

Units none

Default 0

Remarks This parameter controls which MACRO nodes PMAC performs automatic copying into and out of the auxiliary registers. Enabling this function for a node is required to use the auxiliary register as the flag register for a motor.

I1000 is a 16-bit variable. Bits 0 to 15 control the enabling of this copying function for MACRO nodes 0 to 15, respectively. A bit value of 1 means the copying function is enabled; a bit value of 0 means the copying function is disabled.

If the copying function is enabled for Node n (where n = 0 to F hex or 0 to 15 decimal), during each background "housekeeping" software cycle, PMAC copies the contents of Y:$0F7n to the Node n auxiliary write register, and copies the contents of the Node n auxiliary read register into X:$0F7n.

The copying function enabled by I1000 permits PLC and on-line-command auxiliary read and write functions plus use of the auxiliary registers for command and status flags.

See Also MACRO Setup
I-Variables Ix25, I995, I996
On-line commands

I1001 MACRO Ring Check Period

Range 0 .. 255

Units servo cycles

Default 0

Remarks I1001 determines the period for PMAC to evaluate whether there has been a MACRO ring failure. If I1001 is greater than 0, PMAC must receive a sufficient number of "sync node" packets (the packet is specified by I996), and not detect too many ring communications errors, in I1001 servo cycles. If either of these conditions is not met, PMAC will consider there to be a ring fault, and it will disable all of its slave MACRO nodes.

If I1001 is 0 at power-up/reset, PMAC does not perform these checks, even if the MACRO ring is active. To start performing these checks, set I1001 to a value greater than 0, issue a **SAVE** command, then reset the card with a **$$$** command.

If I1001 is greater than 0 at power-up/reset, the check period can be changed immediately by changing the value of I1001; there is no need to SAVE the new value and reset the card to get the new value to take effect.

In PMAC firmware versions V1.16D and older, PMAC performs these checks during its background "housekeeping" cycle, executed once between each scan of each background uncompiled PLC (all compiled background PLCs execute a scan each between each housekeeping cycle). Each cycle, it can detect at most one sync node packet and one communications error. In these firmware versions, the number of sync node packets required in an I1001 check period to continue operation is fixed at 2, and the number of communications errors in an I1001 check period that will cause disabling of operations over the ring is fixed at 2.

In PMAC firmware versions V1.16E and newer, PMAC performs these checks during its "real-time" interrupt (RTI) tasks, every (I8 + 1) servo cycles. Each RTI, it can detect at most one sync node packet and one communications error. In these firmware versions the number of sync node packets required in an I1001 check period to continue operation is set by I1005, and the number of communications errors in an I1001 check period that will cause disabling of operations over the ring is set by I1004.

In all firmware, it is vital that I1001 be set large enough that enough checks of the ring can be executed in the allotted I1001 check period.

With the default servo update of 2.25 kHz (440 usec), an I1001 value of 10 sets the check period at 4.4 msec. An I1001 value of 20 sets the check period at 8.8 msec.

See Also I-variables I996, I1004, I1005

I1002 MACRO Node Protocol Type Control

Range 0 .. $FFFF (0 .. 65,535)

Units none

Default 0

Remarks I1002 controls for each MACRO node (0 – 15) whether the Type 0 or Type 1 MACRO protocol is used on that node. I1002 is a 16-bit value; each bit 0 – 15 controls the protocol type for the MACRO node of the same number. A value of 0 in the bit selects the Type 0 protocol for the matching MACRO node; a value of 1 in the bit selects the Type 1 protocol for the node.

The key difference between Type 0 and Type 1 protocols is in which node register is used for control and status flags. In the Type 0 protocol, the 1st register (24 bits) is used for the flags; in the Type 1 protocol, the 4th registers (16 bits) is used for the flags. The bits of I1002 must be set properly for any node whose auxiliary flag function is enabled by I1000.

The Type 0 protocol is generally used for older single-node MACRO devices, such as the Performance Controls FLX Drive. The Type 1 protocol is generally used for multi-node MACRO devices, such as Delta Tau's MACRO Station (MACRO Stack or UMAC MACRO). With the Delta Tau MACRO Station, I1002 is generally set to the same value as I1000.

See Also I-variables Ix25, I1000

I1003 MACRO Type 1 Master/Slave Communications Timeout

Range 0 .. 255

Units servo cycles

Default 0

Remarks I1003 permits the enabling of MACRO Type 1 master-slave auxiliary communications using Node 15, which are executed with the **MS**, **MSR**, and **MSW** commands. If I1003 is set to 0, these communications are disabled. If I1003 is set to a value greater than 0, these communications are enabled, and the value of I1003 sets the "timeout" value for the auxiliary response, in PMAC servo cycles.

If PMAC has not received a response to the MACRO auxiliary communications command within I1003 servo cycles, it will stop waiting and register a "MACRO Auxiliary Communications Error", setting Bit 5 of global status register X:$000006. A value of 32 for I1003 is suggested.

Bit 15 of I1000 must be set to 0 to disable Node 15's Type 0 (node-specific) auxiliary communications if II1003 is greater than 0. If a value of I1003 greater than 0 has been saved into PMAC's non-volatile memory, then at subsequent power-up/resets, bit 15 of I1000 is automatically forced to 0 by PMAC firmware, regardless of the value saved for I1000.

See Also I-Variable I1000
On-line commands **MACROSLV**, **MACROSLVREAD**, **MACROSLVWRITE**
Program commands **MACROSLVREAD**, **MACROSLVWRITE**

I1004 MACRO Ring Error Shutdown Count

Range 0 .. 255

Units MACRO ring errors

Default 2

Remarks I1004 determines the number of MACRO communications errors detected in one ring check period that will cause the PMAC to conclude that the ring operation is defective. This check is only performed if the I1001 MACRO ring check period parameter is set greater than 0 at power-up/reset. In this case, if PMAC detects I1004 or greater MACRO communications errors in I1001 servo cycles, it will kill all of its motors.

PMAC can detect one ring communications error per real-time interrupt (I8+1 servo cycles) even if more than one error has occurred. Valid settings of I1004 are less than I1001/(I8+1). Regardless of the setting of I1004, if a ring error is detected on every check during the period, a "ring fault" is declared.

PMAC can detect four types of MACRO communications errors: byte "violation" errors, packet checksum errors, packet overrun errors, and packet underrun errors.

If I1004 is set to 0 at power-on/reset, the PMAC will automatically set it to 2.

Before I1004 was implemented, a fixed value of 2 ring errors was used.

See Also I-Variables I8, I995, I1001, I1004

I1005 MACRO Ring Sync Packet Shutdown Count

Range 0 .. 65,535

Units MACRO sync packets

Default 4

Remarks I1005 determines the minimum number of MACRO "sync node" communications packets ("sync packets") that must be detected in one ring check period for PMAC to conclude the the ring is operating properly and permit normal machine operation to continue. This check is only performed if the I1001 MACRO ring check period parameter is set greater than 0. In this case, if PMAC detects fewer than I1005 MACRO sync packets in I1001 servo cycles, it will cause the PMAC to "kill" all of its motors.

PMAC can detect one MACRO sync packet per real-time interrupt (I8+1 servo cycles) even if more than one sync packet has been received in that period. Valid settings of I1005 are less than or equal to I1001/(I8+1). Setting I1005 to a value greater than I1001/(I8+1) means that PMAC will never receive enough sync packets and will always disable its slave stations on the ring.

The node number *n*, 0 to 15, of the sync packet is determined by bits 16 to 19 (the second hex digit) of I996. This node *n* must be activated by setting bit *n* of I996 to 1; otherwise, PMAC will immediately detect a ring communications error.

If I1005 is set to 0 at power-on/reset, the PMAC will automatically set it to 2.

Before I1005 was implemented, a fixed value of 2 sync packets was used.

See Also I-Variables I8, I995, I996 I1001, I1005

I1010 Resolver Excitation Phase Offset {Geo PMAC only}

Range 0 – 255

Units 1/256 cycle

Default 0

Remarks I1010 specifies the phase (time) offset for the AC excitation created by the Geo PMAC for resolvers. The optimum setting of I1010 depends on the L/R time constant of the resolver circuit. I1010 should be set interactively so as to maximize the magnitudes of the feedback ADC values (Y:$FF00 and Y:$FF01 for Resolver 1; Y:$FF20 and Y:$FF21 for Resolver 2).

I1010 is only used if the Geo PMAC's Feedback Option 1 for analog position feedback is ordered.

I1011 Resolver Excitation Gain {Geo PMAC only}

Range 0 – 3

Units Gain-1

Default 0

Remarks I1011 specifies the gain of the AC excitation output created by the Geo PMAC for resolvers, with the gain equal to (I1011 + 1). With a gain of 1, the nominal AC output has peak voltages of +/-2.5V. The following table lists the possible values of I1011 and the nominal output magnitudes they produce:

I1011	Excitation Mag.
0	+/-2.5V
1	+/-5.0V
2	+/-7.5V
3	+/-10.0V

I1011 is only used if the Geo PMAC's Feedback Option 1 for analog position feedback is ordered.

I1012 Resolver Excitation Frequency Divider {Geo PMAC only}

Range 0 – 3

Units none

Default 0

Remarks I1012 specifies the frequency of the AC excitation output created by the Geo PMAC for resolvers as a function of the phase clock frequency set by I900 and I901. The following table lists the possible values of I1012 and the excitation frequencies they produce:

I1012	Excitation Freq.
0	PhaseFreq
1	PhaseFreq/2
2	PhaseFreq/4
3	PhaseFreq/6

I1012 is only used if the Geo PMAC's Feedback Option 1 for analog position feedback is ordered.

I1013 Motor Temperature Check Enable {Geo PMAC only}

Range 0 – 3

Units none

Default 0

Remarks I1013 controls whether the motor temperature check function is enabled for the motor(s) connected to the Geo PMAC. I1013 is a 2-bit value: bit 0 controls whether the temperature check function is enabled for Motor 1, and bit 1 controls whether the temperature check function is enabled for Motor 2. The following table shows the four possible values of I1013 and the functions they produce:

I1013	Motors to Check Temperature
0	Neither
1	Motor 1 only
2	Motor 2 only
3	Motors 1 & 2

If the Geo PMAC is checking temperature for the motor, the motor thermal sensor must be connected to pin 23 of the main encoder connector for the motor.

I1015 SSI Clock Frequency Control {New, Geo PMAC only}

Range 0 – 3

Units none

Default 0

Remarks I1015 specifies the frequency of the digital clock output for the SSI-encoder interfaces on the Geop PMAC. The following table lists the possible values of I1015 and the clock frequencies they produce:

I1015	SSI Clock Freq.
0	153.6 kHz
1	307.2 kHz
2	614.4 kHz
3	1.2288 MHz

I1015 is only used if the Geo PMAC's Feedback Option 2 for absolute position feedback is ordered.

I1016 SSI Channel 1 Mode Control {Geo PMAC only}

Range 0 – 3

Units None

Default 3

Remarks I1016 specifies the mode for interpreting data from the first SSI-encoder interface on a Geo PMAC. The following table lists the possible values of I1016 and the data formats they cause the Geo PMAC to expect:

I1016	SSI Clock Freq.
0	(Reserved)
1	(Reserved)
2	Numeric binary
3	Gray code

I1016 is only used if the Geo PMAC's Feedback Option 2 for absolute position feedback is ordered.

I1017 SSI Channel 1 Word Length Control {Geo PMAC only}

Range 0 – 3

Units none

Default 3

Remarks I1017 specifies the word length in bits from the first SSI-encoder interface on a Geo PMAC. The following table lists the possible values of I1017 and the word lengths they cause the Geo PMAC to request:

I1017	Word Length
0	12 bits
1	16 bits
2	20 bits
3	24 bits

I1017 is only used if the Geo PMAC's Feedback Option 2 for absolute position feedback is ordered.

I1018 SSI Channel 2 Mode Control {Geo PMAC only}

Range 0 – 3

Units None

Default 3

Remarks I1018 specifies the mode for interpreting data from the second SSI-encoder interface on a Geo PMAC. The following table lists the possible values of I1018 and the data formats they cause the Geo PMAC to expect:

I1018	SSI Clock Freq.
0	(Reserved)
1	(Reserved)
2	Numeric binary
3	Gray code

I1018 is only used if the Geo PMAC's Feedback Option 2 for absolute position feedback is ordered.

I1019 SSI Channel 2 Word Length Control {Geo PMAC only}

Range 0 – 3

Units None

Default 3

Remarks I1019 specifies the word length in bits from the second SSI-encoder interface on a Geo PMAC. The following table lists the possible values of I1019 and the word lengths they cause the Geo PMAC to request:

I1019	Word Length
0	12 bits
1	16 bits
2	20 bits
3	24 bits

I1019 is only used if the Geo PMAC's Feedback Option 2 for absolute position feedback is ordered.

I1020 Lookahead Length {Option 6L firmware only}

Range 0 – 65,535

Units I13 segmentation periods

Default 0

Remarks I1020 controls the enabling of the lookahead buffering function for the coordinate system that has a defined lookahead buffer, and if enabled, determines how far ahead the buffer will look ahead.

If I1020 is set to 0 (the default), the buffered lookahead function is not used, even if a lookahead buffer has been defined.

If I1020 is set to 1, points are stored in the lookahead buffer as they are calculated, but no lookahead velocity or acceleration-limiting calculations are done. The stored points can then be used to back up along the path as necessary.

If I1020 is set to a value greater than 1, PMAC will look I1020 segments ahead on LINEAR and CIRCLE mode moves, provided that the PMAC is in segmentation mode (I13 > 0) and a lookahead buffer has been defined. The lookahead algorithm can extend the time for each segment in the buffer as needed to keep velocities under the Ix16 limits and the accelerations under the Ix17 limits.

For proper lookahead control, I1020 must be set to a value large enough so that PMAC looks ahead far enough that it can create a controlled stop from the maximum speed within the acceleration limit. This required stopping time for a motor can be expressed as:

$$StopTime = \frac{V_{max}}{A_{max}} = \frac{Ix16}{Ix17}$$

All motors in the coordinate system should be evaluated to see which motor has the longest stopping time. This motor's stopping time will be used to compute I1020.

The average speed during this stopping time is $V_{max}/2$, so as the moves enter the lookahead algorithm at V_{max} (the worst case), the required time to look ahead is $StopTime/2$. Therefore, the required number of segments always corrected in the lookahead buffer can

be expressed as:

$$SegmentsAhead = \frac{StopTime(msec)/2}{SegTime(msec/seg)} = \frac{Ix16}{2*Ix17*I13}$$

Because PMAC does not completely correct the lookahead buffer as each segment is added, the lookahead distance specified by I1020 must be slightly larger than this. The formula for the minimum value of I1020 that guarantees sufficient lookahead for the stopping distance is:

$$I1020 = \frac{4}{3} * SegmentsAhead$$

If a fractional value results, round up to the next integer. A value of I1020 less than this amount will not result in velocity or acceleration limits being violated; however, the algorithm will not permit maximum velocity to be reached, even if programmed.

I1020 should not be set greater than the number of segments reserved in the **DEFINE LOOKAHEAD** command. If the lookahead algorithm runs out of buffer space, PMAC will automatically reduce I1020 to reflect the amount of space that is available.

Example

The axes in a system have a maximum speed of 24,000 mm/min, or 400 mm/sec (900 in/min or 15 in/sec). They have a maximum acceleration of 0.1g or 1000 mm/sec^2 (40 in/sec^2), and a count resolution of 1μm. A maximum block rate of 200 blocks/sec is desired, so I13 is set to 5 msec. The parameters can be computed as:

- Ix16 = 400 mm/sec * 0.001 sec/msec * 1000 cts/mm = 400 cts/msec
- Ix17 = 1000 mm/sec^2 * 0.001^2 sec^2/msec^2 * 1000 cts/mm = 1.0 cts/msec^2
- I1020 = [4/3] * [400 cts/msec / (2 * 1.0 cts/msec^2 *5 msec/seg)] = 54 segments

I1021 Lookahead State Control {Option 6L Firmware Only}

Range 0 – 15

Units none

Default 0

Remarks I1021 permits direct control of the state of lookahead execution, without going through PMAC's background command interpreter. This is useful for applications such as wire EDM, which can require very quick stops and reversals.

- Setting I1021 to 4 is the equivalent of issuing the **** quick-stop command
- Setting I1021 to 7 is the equivalent of issuing the **<** back-up command
- Setting I1021 to 6 is the equivalent of resuming forward motion with the **>** resume forward command.

If you are monitoring I1021 at other times, you will see that the "4's" bit is cleared after the command has been processed. Therefore, you will see the following values:

- I1021 = 0 when stopped with a quick-stop command
- I1021 = 3 when running reversed in lookahead
- I1021 = 2 when running forward in lookahead

Note:

In preliminary versions of the special PMAC lookahead firmware, I1021 served a different function. That variable value is now a constant value (3) set by the firmware.

PMAC ON-LINE COMMAND SPECIFICATION

<CONTROL-A>

Function Abort all programs and moves.

Scope Global

Syntax ASCII Value 1D; $01

Remarks This command aborts all motion programs and stops all non-program moves on the card. It also brings any disabled or open-loop motors to an enabled zero-velocity closed-loop state. Each motor will decelerate at a rate defined by its own motor I-variable I*x*15. However, a multi-axis system may not stay on its programmed path during this deceleration.

A **<CTRL-A>** stop to a program is not meant to be recovered from gracefully, because the axes will in general not stop at a programmed point. The next programmed move will not behave properly unless a **PMATCH** command is given or I14 is set to 1 (these cause PMAC to use the aborted position as the move start position). Alternately, an on-line **J** command may be issued to each motor to cause it to move to the end point that was programmed when the abort occurred. Then the program(s) can be resumed with an **R** (run) command.

To stop a motion sequence in a manner that can be recovered from easily, use instead the Quit (**Q** or **<CTRL-Q>**) or the Hold (**H** or **<CTRL-O>**) command.

When PMAC is set up to power on with all motors killed (I*x*80 = 0), this command can be used to enable all of the motors (provided that they are not commutated by PMAC – in that case, each motor should be enabled with the **$** command).

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also Stop Commands (Making Your Application Safe)
On-line commands **A**, **$**, **J=**, **PMATCH**, **H**, **<CTRL-O>**, **Q**, **<CTRL-Q>**
I-variables **I14**, **I*x*15**, **I*x*80**.

<CONTROL-B>

Function Report status word for all motors.

Scope Global

Syntax ASCII Value 2D; $02

Remarks This command causes PMAC to report the status words for all of the motors to the host in hexadecimal ASCII form, 12 characters per motor starting with motor #1, with the characters for each motor separated by spaces. The characters reported for each motor are the same as if the **?** command had been issued for that motor.

The detailed meanings of the individual status bits are shown under the **?** command description.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (**@n**).

Example
```
<CTRL-B>
812000804001 812000804001 812000A04001 812000B04001 050000000000
050000000000 050000000000 050000000000<CR>
```

See Also On-line commands **<CTRL-C>**, **<CTRL-G>**, **?**, **@*n***
Memory-map registers X:$003D, X:$0079, etc., Y:$0814, Y:$08D4;
Suggested M-Variable definitions Mx30-Mx45.

<CONTROL-C>

Function Report all coordinate system status words

Scope Global

Syntax ASCII Value 3D, $03

Remarks This command causes PMAC to report the status words for all of the coordinate systems to the host in hexadecimal ASCII form, 12 characters per coordinate system starting with coordinate system 1, with the characters for each coordinate system separated by spaces. The characters reported for each coordinate system are the same as if the **??** command had been issued for that coordinate system.

The detailed meanings of the individual status bits are shown under the **??** command description.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (by the **@n** command).

Example

```
<CTRL-C>
A80020020000 A80020020000 A80020020000 A80020020000 A80020000000
A80020000000 A80020000000 A80020000000<CR>
```

See Also On-line commands **<CTRL-B>**, **<CTRL-G>**, **??**;
Memory-map registers Y:$0817, Y:$08D7, etc., X:$0818, X:$08D8, etc.;
Suggested M-variable definitions Mx80-Mx90.

<CONTROL-D>

Function Disable all PLC programs.

Scope Global

Syntax ASCII Value 4D; $04

Remarks This command causes all PLC programs to be disabled (i.e. stop executing). This is the equivalent of **DISABLE PLC 0..31** and **DISABLE PLCC 0..31**. It is especially useful if a **CMD** or **SEND** statement in a PLC has run amok.

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

Example

```
TRIGGER FOUND
TRIGTRIGER FOTRIGGER FOUND
TRTRIGTRIGGER FOUND        (Out-of-control SEND message from PLC)
<CTRL-D>.....              (Command to disable the PLCs)
                           (No more messages; can now edit PLC)
```

See Also On-line commands **DISABLE PLC**, **ENABLE PLC**, **DISABLE PLCC**, **ENABLE PLCC**, **OPEN PLC**
Program commands **DISABLE PLC**, **ENABLE PLC**, **DISABLE PLCC**, **ENABLE PLCC**, **COMMAND**, **SEND**

<CONTROL-F>

Function Report following errors for all motors.

Scope Global

Syntax ASCII Value 6D; $06

Remarks This command causes PMAC to report the following errors of all motors to the host. The errors are reported in an ASCII string, each error scaled in counts, rounded to the nearest tenth of a count. A space character is returned between the reported error for each motor.

Refer to the on-line **F** command for more detail as to how the following error is calculated.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (by the **@n** command).

Example
```
<CTRL-F>
0.5 7.2 -38.3 1.7 0 0 0 0<CR>
```

See Also I-variables I*x*11, I*x*12
On-line commands F, <CTRL-P>, <CTRL-V>

<CONTROL-G>

Function Report global status word.

Scope Global

Syntax ASCII Value 7D; $07

Remarks This command causes PMAC to report the global status words to the host in hexadecimal ASCII form, using 12 characters. The characters sent are the same as if the **???** command had been sent, although no command acknowledgement character (<ACK> or <LF>, depending on I3) is sent at the end of the response.

The detailed meanings of the individual status bits are shown under the **???** command description.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (by the **@n** command).

Example
```
<CTRL-G>
003000400000<CR>
```

See Also On-line commands **<CTRL-B>**, **<CTRL-C>**, **???**
Memory-map registers X:$0003, Y:$0003.

<CONTROL-H>

Function Erase last character.

Scope Global

Syntax ASCII Value 8D; $08 (**<BACKSPACE>**).

Remarks This character, usually entered by typing the **<BACKSPACE>** key when talking to PMAC in terminal mode, causes the most recently entered character in PMAC's command-line-receive buffer to be erased.

See Also Talking to PMAC
On-line command **<CTRL-O>** (Feed Hold All)

<CONTROL-I>

Function Repeat last command line.

Scope Global

Syntax ASCII Value 9D; $09 (**<TAB>**).

Remarks This character, sometimes entered by typing the **<TAB>** key, causes the most recently sent alphanumeric command line to PMAC to be re-commanded. It provides a convenient way to quicken a repetitive task, particularly when working interactively with PMAC in terminal mode. Other control-character commands cannot be repeated with this command.

> ***Note:***
>
> Internally generated commands from **CMD "{command}"** statements in motion and PLC programs overwrite the last executed command from the host, and so can alter the action of this character.

> ***Note:***
>
> Most versions of the PMAC Executive Program "trap" a **<CTRL-I>** or **<TAB>** for their own purposes, and do not send it on to PMAC, even when in terminal mode

Example This example shows how the tab key can be used to look for some event:

```
PC<CR>
P1:10<CR>
<TAB>
P1:10<CR>
<TAB>
P1:10<CR>
<TAB>
P1:11<CR>
```

See Also On-line command **<CONTROL-Y>**.

<CONTROL-K>

Function Kill all motors.

Scope Global

Syntax ASCII Value 11D; $0B

Remarks This command kills all motor outputs by opening the servo loop, commanding zero output, and taking the amplifier enable signal (AENA*n*) *false* (polarity is determined by jumper E17) for all motors on the card. If any motion programs are running, they will automatically be aborted. (For the motor-specific **K** (kill) command, if the motor is in a coordinate system that is executing a motion program, the program execution must be stopped with either an **A** (abort) or **Q** (quit) command before PMAC will accept the **K** command.)

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also On-line commands **K**, **A**, **<CONTROL-A>**.

<CONTROL-L>

Function Close open rotary buffer.

Scope Global

Syntax ASCII Value 12D; $0C

Remarks This character causes PMAC to close the open rotary program buffer on PMAC. It is exactly equivalent in effect to the **CLOSE** command, but it is faster to send. The primary use of

<CTRL-L> is when the rotary buffer needs to be opened and closed repeatedly. After closing the rotary buffer, there is no chance that an on-line command can be mistaken for a buffer command.

Example

```
<CTRL-U>                    ; Open rotary buffer
X10 Y20 F5 M3               ; Put program line in buffer
X30 Y40 F5....              ; Put program line in buffer
<CTRL-L>                    ; Close rotary buffer
M1                          ; On-line command for value of M1
1                           ; PMAC responds
```

See Also Rotary Motion Program Buffers (Writing a Motion Program)
On-line commands **<CTRL-U>**, **OPEN ROT**, **CLOSE**

<CONTROL-M>

Function Enter command line.

Scope Global

Syntax ASCII Value 13D; $0D (**<CR>**)

Remarks This character, commonly known as **<CR>** (carriage return), causes the alphanumeric characters in the PMAC's command-line-receive buffer to be interpreted and acted upon. (Control-character commands do not require a **<CR>** character to execute.)

> ***Note:***
>
> For multiple PMACs daisy-chained together on a serial interface, this will act on all cards simultaneously, not just the software-addressed card. For simultaneous action on multiple cards, it is best to load up the command-line-receive buffers on all cards before issuing the <CR> character.

Example

```
#1J+<CR>
P1<CR>
@0&1B1R@1&1B7R<CR>          (This causes card 0 on the serial daisy-chain to have its CS 1
                            execute PROG 1 and card 1 to have its CS 1 execute
                            PROG 7 simultaneously.)
```

See Also Talking to PMAC

<CONTROL-N>

Function Report command line checksum.

Scope Global

Syntax ASCII Value 14D; $0E

Remarks This character causes PMAC to calculate and report the checksum of the alphanumeric characters of the present command lines (i.e. since the most recent carriage-return character).

As typically used, the host computer would send the entire command line up to, but not including, the carriage return. It would then send the **<CTRL-N>** character, and PMAC would return the checksum value. If this value agreed with the host's internally calculated checksum value, the host would then send the **<CR>** and PMAC would execute the command line. If the values did not agree, the host would send a **<CTRL-X>** command to erase the command line, then resend the line, repeating the process.

> ***Note:***
>
> The PMAC Executive Program terminal mode will not display the checksum values resulting from a **<CTRL-N>** command.

Example

With I4=1 and I3=2:

```
Host sends:..... J+<CTRL-N>
PMAC sends:..... <117dec>          (117=74[J] + 43[+]; correct)
Host sends:..... <CR>
PMAC sends:..... <ACK><117dec>     (handshake and checksum again)
Host sends:..... J/<CTRL-N>
PMAC sends:..... <122dec>          (122 != 74[J] +47[/]; incorrect)
Host sends:..... <CTRL-X>          (Erase the incorrect command)
 ..... J/<CTRL-N>                        (Send the command again)
```

PMAC sends:..... <121dec> (121 = 74[J] + 47[/]; correct)
Host sends:..... **<CR>**
PMAC sends:..... <ACK><121dec> (handshake and checksum again)

See Also Communications Checksum (Writing a Host Communications Program)
I-variables I3, I4
On-line commands **<CTRL-M>** (**<CR>**), **<CTRL-X>**

<CONTROL-O>

Function Feed hold on all coordinate systems.

Scope Global

Syntax ASCII Value 15D; $0F

Remarks This command causes all coordinate systems in PMAC to undergo a feed hold. A feed hold is much like a **%0** command where the coordinate system is brought to a stop without deviating from the path it was following, even around curves. However, with a feed hold, the coordinate system slows down at a slew rate determined by I*x*95, and can be started up again with an **R** (run)command. The system then speeds up at the rate determined by I*x*95, until it reaches the desired value (from internal *or* external timebase). From then on, any timebase changes occur at a rate determined by I*x*94.

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

On a flash memory PMAC that is in bootstrap mode (powered up with E51 ON), the **<CTRL-O>** command puts PMAC into its firmware reload command. All subsequent characters sent to PMAC are interpreted as bytes of machine code for PMAC's operational firmware, overwriting the existing operational firmware in flash memory.

See Also Resetting PMAC (Talking to PMAC)
I-variables I52, I*x*94, I*x*95
On-line commands **<CTRL-H>** (backspace) **H** (feedhold), **R** (run), **%** (feedrate override), **** (program hold).
Jumper E51

<CONTROL-P>

Function Report positions of all motors.

Scope Global

Syntax ASCII Value 16D; $10

Remarks This command causes the positions of all motors to be reported to the host. The positions are reported as an ASCII string, scaled in counts, rounded to the nearest tenth of a count, with a space character in between each motor's position.

The position window in the PMAC Executive program works by repeatedly sending the **<CTRL-P>** command and rearranging the response into the window.

PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register.

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (by the **@n** command).

Example **<CTRL-P>**
9999.5 10001.2 5.7 -2.1 0 0 0 0<CR>

See Also On-line commands **P**, **<CTRL-V>**, **<CTRL-F>**.

<CONTROL-Q>

Function Quit all executing motion programs.

Scope Global

Syntax ASCII Value 17D; $11

Remarks This command causes any and all motion programs running in any coordinate system to stop executing after the moves that have already been calculated are finished. Program execution may be resumed from this point with the **R** (run) or **S** (step) commands.

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also On-line commands **<CTRL-A>**, **<CTRL-K>**, **<CTRL-O>**, **<CTRL-R>**, **<CTRL-S>**, **Q**
Motion-program command **STOP**.

<CONTROL-R>

Function Begin execution of motion programs in all coordinate systems.

Scope Global

Syntax ASCII Value 18D; $12

Remarks This command is the equivalent of issuing the **R** (run) command to all coordinate systems in PMAC. Each active coordinate system (i.e. one that has at least one motor assigned to it) that is to run a program must already be pointing to a motion program (initially this is done with a **B{*prog num*}** command).

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

For a flash memory PMAC that is in bootstrap mode (powered up with E51 ON), the **<CTRL-R>** command puts PMAC into normal operational mode, but with factory default I-variables, conversion table settings, and VME/DPRAM addresses.

Example
```
&1B1&2B500<CR>
<CTRL-R>
```

See Also Executing a Motion Program (Writing a Motion Program)
Resetting PMAC (Talking to PMAC)
On-line commands **R**, **B**.
Jumper E51

<CONTROL-S>

Function Step working motion programs in all coordinate systems.

Scope Global

Syntax ASCII Value 19D; $13

Remarks This command is the equivalent of issuing an **S** (step) command to all of the coordinate systems in PMAC. Each active coordinate system (i.e. one that has at least one motor assigned to it) that is to run a program must already be pointing to a motion program (initially this is done with a **B{*prog num*}** command).

A program that is not running will execute all lines down to and including the next motion command (move or dwell), or if it encounters a **BLOCKSTART** command first, all lines down to and including the next **BLOCKSTOP** command.

If a program is already running in continuous execution mode (from an **R** (run) command), an **S** command will put the program in single-step mode, stopping execution after the next motion command). In this situation, it has exactly the same effect as a **Q** (quit) command.

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also On-line commands **<CTRL-A>**, **<CTRL-O>**, **<CTRL-Q>**, **<CTRL-R>**, **A**, **H**, **O**, **Q**, **R**, **S**;
Motion-program commands **BLOCKSTART**, **BLOCKSTOP**, **STOP**.
Control-panel port (JPAN) input STEP/.

<CONTROL-T>

Function Toggle serial port half/full duplex mode.

Scope Global

Syntax ASCII Value 20D; $14

Remarks This causes serial port communications to toggle between half duplex (PMAC will not echo character back to host) and full duplex (PMAC will echo character back to host). The power-up default is half duplex.

This command is invalid when multiple PMACs are daisy-chained on a single serial interface.

See Also Data Integrity Checks (Writing a Host Communications Program)
On-line command **<CTRL-Z>**.

<CONTROL-U>

Function Open rotary program buffer(s).

Scope Global

Syntax ASCII Value 21D; $15

Remarks This character causes PMAC to open all existing rotary motion program buffers for entry. It is exactly equivalent in effect to the **OPEN ROTARY** command, but it is faster to send. Along with the **<CTRL-L>** command, it permits rapid opening and closing of the rotary buffer, so that on-line commands can be sent with the buffer closed without chance that they will be mistaken for buffer commands.

Example

```
<CTRL-L> .....      ; Close rotary buffer
M1 .....            ; On-line command for value of M1
```

```
1.....                  ; PMAC responds
<CTRL-U>.....           ; Open rotary buffer
X10 Y20 F5 M3           ; Put program line in buffer
X30 Y40 F5              ; Put program line in buffer
```

See Also Rotary Motion Program Buffers (Writing a Motion Program)
On-line commands **<CTRL-L>**, **OPEN ROT**, **CLOSE**

<CONTROL-V>

Function Report velocity of all motors.

Scope Global

Syntax ASCII Value 22D; $16

Remarks This command causes PMAC to report the velocities of all motors to the host. The velocity units are in encoder counts per servo cycle, rounded to the nearest tenth. The <F7> velocity window in the PMAC Executive program works by repeatedly issuing the **<CTRL-V>** command and displaying the response on the screen.

To scale these values into counts/msec, multiply the response by 8,388,608*(Ix60+1)/I10 (servo cycles/msec).

> ***Note:***
>
> The velocity values reported here are obtained by subtracting positions of consecutive servo cycles. As such, they can be very noisy. For purposes of display, it is probably better to use averaged velocity values held in registers Y:$082A, Y:$08EA, etc., accessed with M-variables

For multiple cards on a single serial daisy-chain, this command affects only the card currently addressed in software (**@n**).

See Also I-variable I10, Ix60
On-line commands **<CTRL-F>**, **<CTRL-P>**, **V**
Memory registers X:$0033, X:$006F, etc.
Suggested M-variable definitions Mx66

<CONTROL-X>

Function Cancel in-process communications.

Scope Global

Syntax ASCII Value 24D; $18

Remarks This command causes the PMAC to stop sending any messages that it had started to send, even multi-line messages. This also causes PMAC to empty the command queue from the host, so it will erase any partially sent commands.

It can be useful to send this before sending a query command for which you are expecting an exact response format, if you are not sure what PMAC has been doing before, because it makes sure nothing else comes through before the expected response. As such, it is often the first character sent to PMAC from the host when trying to establish initial communications. In addition, many Delta Tau communications routines start by sending a **<CTRL-X>** command to ensure that there is no previously pending response that could confuse the host software.

If I63 is set to 0, there is no acknowledgment of the completion of the **<CTRL-X>** command. If I63 is set to 1, PMAC acknowledges the completion of the command with a **<CTRL-X>** to the host, permitting the host to know that it is safe to send the next command. PCOMM32 versions 10.21 and newer can take advantage of this feature to improve the speed of communications.

> ***Note:***
>
> This command empties the command queue in PMAC RAM, but it cannot erase the 1 or 2 characters already in the response port. A robust algorithm for clearing responses would include 2 character-read commands that can time-out if necessary.

For multiple cards on a single serial daisy-chain, this command affects all cards on the chain, regardless of the current software addressing.

See Also On-line commands **<CTRL-H>**, **<CTRL-Z>**

<CONTROL-Y>

Function Report last command line.

Scope Global

Syntax ASCII Value 25D; $19

Remarks This causes PMAC to report the last command line to the host (with no trailing <*CR*>) and to re-enter the line into the command queue ready to execute upon the next receipt of <*CR*>. This allows a user communicating with PMAC in terminal mode to recall the last command and to be able to edit it with the backspace and typing in desired changes. The command will be re-executed when the host sends a <*CR*>.

> ***Note:***
>
> Internally generated commands from **CMD "{command}"** statements in motion and PLC programs overwrite the last executed command from the host, and so can alter the action of this character.

Example

P123=5<CR> .	;Set the first value
P124=7<CR> ...	;Set the second value
P123<CR>	;Query the first value
5	;PMAC responds with value
<CTRL-Y>	;Tell PMAC to report last command
P123	;PMAC reports last command
<BACKSPACE>4<CR>	;Modify to **P124** and send
7	;PMAC tells value of P124

See Also On-line command **<CONTROL-I>**.

<CONTROL-Z>

Function Set PMAC in serial port communications mode.

Scope Global

Syntax ASCII Value 26D; $1A

Remarks This command causes PMAC's serial port to become the active communications output port. All PMAC responses directed to the host will be sent over the serial port. This mode will continue until a command is received over the bus (parallel) port, which will make the bus port the active communications output port. PMAC powers up/resets with the serial port the

active port.

If you are trying to establish communications with PMAC over the serial port, it is a good idea to send this character before any query commands to make sure PMAC will try to respond over the serial port.

Regardless of which is the active output port, PMAC can accept commands over either port. It is the user's responsibility not to garble commands by simultaneously commanding over both ports.

Example

Serial host sends:	**P1**
PMAC responds to serial port:	*12*
Bus host sends: ..	**P1=P1+1**
Serial host sends:	**P1**
PMAC responds to *bus port:*	*13*
(Serial host gets no response)	
Serial host sends:	**<CTRL-Z>P1**
PMAC responds to serial port:	*13*

See Also On-line commands **<CTRL-T>**, **<CTRL-X>**;
Jumpers E44-E47.

#

Function Report currently addressed motor

Scope Global

Syntax **#** Ask PMAC which motor is addressed

Remarks This causes PMAC to return the number of the motor currently addressed by the host – the one that acts upon motor-specific commands from the host.

> ***Note:***
>
> A different motor may be hardware selected from the control panel port for motor-specific control panel inputs, and that different motors may be addressed from programs within PMAC for **COMMAND** statements.

Example

```
#
2          PMAC reports that motor 2 is addressed
```

See Also Control-Panel Port Inputs (Connecting PMAC to the Machine)
On-line commands **#{constant}**, **&**, **&{constant}**, **@{constant}**
Program commands **ADDRESS**, **COMMAND**

#{constant}

Function Address a motor.

Scope Global

Syntax **#{constant)**
where:

- **{constant}** is an integer from 1 to 8, representing the number of the motor to be addressed

Remarks This command makes the motor specified by **{constant}** the addressed motor (the one on which on-line motor commands will act). The addressing is modal, so all further motor-specific commands will affect this motor until a different motor is addressed. At power-

up/reset, Motor 1 is addressed.

> ***Note:***
>
> A different motor may simultaneously be hardware selected from the control panel port for motor-specific control panel inputs, and that different motors may be addressed from programs within PMAC for **COMMAND** statements.

Example

```
#1J+.....        ;Command Motor 1 to jog positive
J- .....         ;Command Motor 1 to jog negative
#2J+.....        ;Command Motor 2 to jog positive
J/               ;Command Motor 2 to stop jogging
```

See Also Control-Panel Port Inputs (Connecting PMAC to the Machine)
Addressing commands (Talking to PMAC)
Program commands **COMMAND**, **ADDRESS**
On-line commands **#**, **&**, **&{constant}**, **@{constant}**

#{constant}->

Function Report the specified motor's coordinate system axis definition.

Scope Coordinate-system specific

Syntax **#{constant**

where:

- **{constant}** is an integer from 1 to 8 representing the number of the motor whose axis definition is requested

Remarks This command causes PMAC to report the current axis definition of the specified motor in the currently addressed coordinate system. If the motor has not been defined to an axis in the currently addressed system, PMAC will return a **0** (even if the motor has been assigned to an axis in another coordinate system). A motor can have an axis definition in only one coordinate system at a time.

Example

```
&1 .....          ; Address Coordinate System 1
#1->.....         ; Request Motor 1 axis definition in C.S. 1
10000X..... ; PMAC responds with axis definition
&2 .....          ; Address Coordinate System 2
#1->.....         ; Request Motor 1 axis definition in C.S. 2
0           ; PMAC shows no definition in this C.S.
```

See Also Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **#{constant}->0**, **#{constant}->{axis definition}**, **UNDEFINE**, **UNDEFINE ALL**.

#{constant}->0

Function Clear axis definition for specified motor.

Scope

Syntax **#{constant}->0**

where:

- **{constant}** is an integer from 1 to 8 representing the number of the motor whose axis definition is to be cleared

Remarks This command clears the axis definition for the specified motor *if* the motor has been defined to an axis in the currently addressed coordinate system. If the motor is defined to an axis in another coordinate system, this command will not be effective. This allows the motor to be

redefined to another axis in this coordinate system or a different coordinate system.

Compare this command to **UNDEFINE**, which erases all the axis definitions in the addressed coordinate system, and to **UNDEFINE ALL**, which erases all the axis definitions in all coordinate systems.

Example This example shows how the command can be used to move a motor from one coordinate system to another:

```
&1 ..................... ; Address C.S. 1
#4->.................... ; Request definition of #4
5000A ..... ; PMAC responds
#4->0 ..... ; Clear definition
&2 ..... ; Address C.S. 2
#4->5000A..... ; Make new definition in C.S. 2
```

See Also Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **UNDEFINE**, **UNDEFINE ALL**, **#{constant}->{axis definition}**.

#{constant}->{axis definition}

Function Assign an axis definition for the specified motor.

Scope Coordinate-system specific

Syntax **#{constant}->{axis definition}**

where:

- **{constant}** is an integer from 1 to 8 representing the number of the motor whose axis definition is to be made;
- **{axis definition}** consists of:
  - 1 to 3 sets of **[{scale factor}]{axis}**, separated by the **+** character, in which:
    - the optional **{scale factor}** is a floating-point constant representing the number of motor counts per axis unit (engineering unit); if none is specified, PMAC assumes a value of 1.0;
    - **{axis}** is a letter (X, Y, Z, A, B, C, U, V, W) representing the axis to which the motor is to be matched;
  - **[+{offset}]** (optional) is a floating-point constant representing the difference between axis zero position and motor zero (home) position, in motor counts; if none is specified, PMAC assumes a value of 0.0

> ***Note:***
>
> No space is allowed between the motor number and the arrow double character.

Remarks This command assigns the specified motor to a set of axes in the addressed coordinate system. It also defines the scaling and starting offset for the axis or axes.

In the vast majority of cases, there is a one-to-one matching between PMAC motors and axes, so this axis definition statement only uses one axis name for the motor.

A scale factor is typically used with the axis character, so that axis moves can be specified in standard units (e.g. millimeters, inches, degrees). This number is what defines what the user units will be for the axis. If no scale factor is specified, a user unit for the axis is one motor count. Occasionally an offset parameter is used to allow the axis zero position to be different

from the motor home position. (This is the starting offset; it can later be changed in several ways, including the **PSET**, **{axis}=**, **ADIS**, and **IDIS** commands).

If the specified motor is currently assigned to an axis in a different coordinate system, PMAC will reject this command (reporting an ERR003 if I6=1 or 3). If the specified motor is currently assigned to an axis in the addressed coordinate system, the old definition will be overwritten by this new one.

To undo a motor's axis definition, address the coordinate system in which it has been defined, and use the command **#{constant}->0**. To clear all of the axis definitions within a coordinate system, address the coordinate system and use the **UNDEFINE** command. To clear all axis definitions in *all* coordinate systems, use **UNDEFINE ALL**.

For more sophisticated systems, two or three cartesian axes may be defined as a linear combination of the same number of motors. This allows coordinate system rotations and orthogonality corrections, among other things. One to three axes may be specified (if only one, it amounts to the simpler definition above). All axes specified in one definition must be from the same triplet set of cartesian axes: XYZ or UVW. If this multi-axis definition is used, a command to move an axis will result in multiple motors moving.

Example

```
#1->X ......          ; User units = counts
#4->2000 A .          ; 2000 counts/user unit
#8->3333.333Z-666.667 ; Non-integers OK
#3->Y .....           ; 2 motors may be assigned to the same axis;
#2->Y .....           ; both motors move when a Y move is given
#1->8660X-5000Y       ;This provides a 30o rotation of X and Y...
#2->5000X+8660Y       ;with 10000 cts/unit – this rotation does
#3->2000Z-6000        ;not involve Z, but it could have
```

This example corrects for a Y axis 1 arc minute out of square:

```
#5->100000X           ;100000 cts/in
#6->-29.1X+100000Y    ;sin and cos of 1/60
```

See Also

Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **#{constant}->**, **#{constant}->0**, **UNDEFINE**, **UNDEFINE ALL**.

$

Function Reset motor

Scope Motor specific

Syntax **$**

Remarks This command causes PMAC to initialize the addressed motor, performing any required commutation phasing and full reading of an absolute position sensor, leaving the motor in a closed-loop zero-velocity state. (For a non-commutated motor with an incremental encoder, the **J/** command may also be used.)

This command is necessary to initialize a PMAC-commutated motor after power-up/reset if Ix80 for the motor is set to 0. If Ix80 is 1, the initialization will be done automatically during the power-up/reset cycle.

This command will not be accepted if the motor is executing a move.

Example

```
I180.....    ; Request value of #1 power-on mode variable
0.....       ; PMAC responds with 0; powers on unphased and killed
$$$.....     ; Reset card; motor is left in killed state
#1$          ; Initialize motor, phasing and reading as necessary
```

See Also Absolute Sensors (Setting Up a Motor)
Power-on Phasing (Setting Up PMAC Commutation)
I-variables Ix10, Ix73, Ix74, Ix75, Ix80, Ix81
On-line commands **$$$**, **J/**

$$$

Function Full card reset.

Scope Global

Syntax **$$$**

Remarks This command causes PMAC to do a full card reset. The effect of **$$$** is equivalent to that of cycling power on PMAC, or taking the INIT/ line low, then high.

With jumper E51 in its default state (OFF for PMAC-PC, -Lite, -VME, ON for PMAC-STD), this command does a standard reset of the PMAC. On PMACs without the Option CPU section (not option 4A, 5A, or 5B), I-variable values, conversion-table settings, and DPRAM and VMEbus addresses stored in permanent memory (EAROM) by the last **SAVE** command are reloaded into active memory (RAM). All information stored in battery backed RAM such as P-variable and Q-variable values, M-variable definitions, and motion and PLC programs are not changed by this command.

On PMACs with the Option CPU section (option 4A, 5A, or 5B), PMAC copies the contents of the flash memory into active memory during a normal reset cycle, overwriting any current contents. This means that anything changed in PMAC's active memory that was not saved to flash memory will be lost. Even the last saved P-variable and Q-variable values, M-variable definitions, and motion and PLC programs are copied from flash to RAM during the reset cycle.

With jumper E51 in non-default state (ON for PMAC-PC, -Lite, -VME, OFF for PMAC-STD), this command does a reset and re-initialization of the PMAC. On PMACs without the Option CPU section (not option 4A, 5A, or 5B), factory default I-variable values, conversion-table settings, and DPRAM and VMEbus addresses stored in the firmware (EPROM) are copied into active memory (RAM). (Values stored in EAROM are not lost; they are simply not used.)

On PMACs with the Option CPU section (option 4A, 5A, or 5B), PMAC enters a special re-initialization mode called "bootstrap mode" that permits the downloading of new firmware (see PMAC PROM SOFTWARE UPDATE LISTING for details of this mode). In this bootstrap mode, there are very few command options. To bypass the download operation in this mode, send a **<CONTROL-R>** character to PMAC. This puts PMAC in the normal operational mode with the existing firmware. Factory default values for I-variables, conversion table settings, and bus addresses for DPRAM and VME are copied from the firmware section of flash memory into active memory. The saved values of these values are not used, but they are still kept in the user section of flash memory.

Because this command immediately causes PMAC to enter its power-up/rest cycle, there is no acknowledging character (**<ACK>** or **<LF>**) returned to the host.

Example

```
I130=60000 ...          ; Change #1 proportional gain
SAVE.....               ; SAVE I-variables to EAROM
I130=80000 ...          ; Change gain again
$$$.....                ; Reset card
I130.....               ; Request value of parameter
60000 .....             ; PMAC reports current value, which is SAVEd value
(Put E51 on)
$$$.....                ; Reset card
I130.....               ; Request value of parameter
2000                    ; PMAC reports current value, which is default
```

See Also

Resetting PMAC (Talking to PMAC)
PMAC PROM SOFTWARE UPDATE LISTING
Control-Panel Port INIT/ Input (Connecting PMAC to the Machine)
On-line command **$$$*****
I-variables I5, Ix80
JPAN Connector Pin 15
Jumper E51.

$$$***

Function Global card reset and re-initialization.

Scope Global

Syntax **$$$*****

Remarks This command performs a full reset of the card and reinitializes the memory. All programs and other buffers are erased. All I-variables, encoder conversion table entries, and VME and DPRAM addressing parameters are returned to their factory defaults. (Previously **SAVE**d values for these parameters are still held in EAROM, and can be brought into active memory with a subsequent **$$$** command). It will also recalculate the firmware checksum reference value and eliminate any PASSWORD that might have been entered.

M-variable definitions, P-variable values, Q-variable values, and axis definitions are not affected by this command. They can be cleared by separate commands (e.g. **M0..1023->***, **P0..1023=0**, **Q0..1023=0**, **UNDEFINE ALL**).

This command is particularly useful if the program buffers have become corrupted. It clears the buffers and buffer pointers so the files can be re-sent to PMAC. Regular backup of parameters and programs to the disk of a host computer is strongly encouraged so this type of recovery is possible. The PMAC Executive program has Save Full PMAC Configuration and Restore Full PMAC Configuration functions to make this process easy.

With jumper E51 in non-default state (ON for PMAC-PC, -Lite, -VME, OFF for PMAC-STD), a PMAC with the Option CPU section (option 4A, 5A, or 5B) enters a special re-initialization mode called "bootstrap mode" when this command is given. This mode permits the downloading of new firmware (see PMAC PROM SOFTWARE UPDATE LISTING for details of this mode). In this mode, there are very few command options. To bypass the download operation in this mode, send a **<CONTROL-R>** character to PMAC. This puts PMAC in the normal operational mode with the existing firmware. Factory default values for I-variables, conversion table settings, and bus addresses for DPRAM and VME are copied from the firmware section of flash memory into active memory. The saved values of these values are not used, but they are still kept in the user section of flash memory.

Example

```
I130=60000 ...      ; Set #1 proportional gain
SAVE.....           ; Save to non-volatile memory
$$$***.....         ; Reset and re-initialize card
I130.....           ; Request value of I130
2000.....           ; PMAC reports current value, which is default
$$$.....            ; Normal reset of card
I130.....           ; Request value of I130
60000               ; PMAC reports current value, which is SAVEd value
```

See Also

On-line command **$$$**;
PMAC PROM Software Update Listing
Jumper E51
PMAC Executive Program Save/Restore Full Configuration.

$*

Function Read motor absolute position

Scope Motor specific

Syntax **$***

Remarks The **$*** command causes PMAC to perform a read of the absolute position for the addressed motor, as defined by Ix10 for the motor. It performs the same actions that are normally performed during the board's power-up/reset cycle.

The **$*** command performs the following actions on the addressed motor:

- The motor is killed (servo loop open, zero command, amplifier disabled).
- If the motor is set up for local hardware encoder position capture by input flags, with bit 16 of Ix03 set to 0 to specify hardware capture, and bit 18 of Ix25 set to 0 to specify local, not MACRO, flag operation (these are default values), the hardware encoder counter for the same channel as the flag register specified by Ix25 is set to 0 (e.g. if Ix25 specifies flags from channel 3, then encoder counter 3 is cleared).
- The motor home complete status bit is cleared.

- The motor position bias register, which contains the difference between motor and axis zero positions, is set to 0.
- If Ix10 for the motor is greater than 0, specifying an absolute position read, the sensor is read as specified by Ix10 to set the motor actual position. The actual position value is set to the sensor value minus the Ix26 "home offset" parameter. Unless the read is determined to be unsuccessful, the motor "home complete" status bit is set to 1.
- If Ix10 for the motor is set to 0, specifying no absolute position read, the motor actual position register is set to 0.
- Because the motor is "killed" the actual position value is automatically copied into the command position register for the motor.
- There are several things to note with regard to this command:
- The motor is left in the "killed" state at the end of execution of this command. To enable the motor, a **$** command should be used if this is a PMAC-commutated motor and a phase reference must be established; otherwise a **J/**, **A**, or **<CTRL-A>** command should be used to enable the motor and close the loop.
- If bit 2 of Ix80 is set to 1, PMAC will not attempt an absolute position read at the board power-on/reset; in this case, the **$*** command must be used to establish the absolute sensor. If bit 2 of Ix80 is set to 0 (the default), PMAC will attempt an absolute position read at the board power-on/reset.
- With Ix10 set to 0, the action of **$*** is very similar to that of the **HOMEZ** command. There are a few significant differences, however:
  - **$*** always kills the motor; **HOMEZ** leaves the servo in its existing state.
  - **$*** sets the present actual position to be zero; **HOMEZ** sets the present commanded position to be zero.
  - **$*** zeros the hardware encoder counter in most cases; **HOMEZ** does not change the hardware encoder counter.

See Also I-variables Ix03, Ix10, Ix25, Ix80, Ix81
On-line commands **$**, **$$$**, **HOMEZ**

%

Function Report the addressed coordinate system's feedrate override value.

Scope Coordinate-system specific

Syntax **%**

Remarks This command causes PMAC to report the present feedrate-override (time-base) value for the currently addressed coordinate system. A value of 100 indicates "'real time"'; i.e. move speeds and times occur as specified.

PMAC will report the value in response to this command, regardless of the source of the value (even if the source is not the **%{constant}** command)

Example

```
%.....                  Request feedrate-override value
100.....                ; PMAC responds: 100 means real time
H.....                  ; Command a feed hold

%.....                  ; Request feedrate-override value
0                       ; PMAC responds: 0 means all movement frozen
```

See Also Time-Base Control (Synchronizing PMAC to External Events)
I-Variables I10, Ix93, Ix94, Ix95
On-line commands **%{constant}**, **H**
Memory map registers X:$0808, X:$08C8, etc.

%{constant}

Function Set the addressed coordinate system's feedrate override value.

Scope Coordinate-system specific

Syntax **%{constant}**
where:

- **{constant}** is a non-negative floating point value specifying the desired feedrate override (time-base) value (100 represents real-time)

Remarks This command specifies the feedrate override value for the currently addressed coordinate system. The rate of change to this newly specified value is determined by coordinate system I-variable Ix94.

I-variable Ix93 for this coordinate system must be set to its default value (which tells to coordinate system to take its time-base value from the % -command register) in order for this command to have any effect.

The maximum % value that PMAC can implement is equal to (2^{23}/I10)*100 or the (servo update rate in kHz)*100. If you specify a value greater than this, PMAC will saturate at this value instead.

To control the time base based on a variable value, assign an M-variable (suggested M197) to the commanded time base register (X:$0806, X:$08C6, etc.), then assign a variable value to the M-variable. The value assigned here should be equal to the desired % value times (I10/100).

Example

```
%0 ..................... ; Command value of 0, stopping motion
%33.333 ..... ; Command 1/3 of real-time speed
%100..... ; Command real-time speed
%500..... ; Command too high a value
%..... ; Request current value
225.88230574 ; PMAC responds; this is max allowed value
M197->X:$0806,24 ; Assign variable to C.S. 1 % command reg.
M197=P1*I10/100 ; Equivalent to &1%(P1)
```

See Also Time-Base Control (Synchronizing PMAC to External Events)
I-Variables I10, Ix93, Ix94, Ix95
On-line commands **%**, **H**
Memory map registers X:$0806, X:$08C6, etc.

&{constant}

Function Address a coordinate system.

Scope Global

Syntax **&{constant}**
where:

- **{constant}** is an integer from 1 to 8, representing the number of the coordinate system to be addressed

Remarks This command makes the coordinate system specified by **{constant}** the addressed coordinate system (the one on which on-line coordinate-system commands will act). The addressing is modal, so all further coordinate-system-specific commands will affect this coordinate system until a different coordinate system is addressed. At power-up/reset, Coordinate System 1 is addressed.

> ***Note:***
>
> A different coordinate system may simultaneously be hardware selected from the control panel port for coordinate-system-specific control panel inputs, and that different coordinate systems may be addressed from programs within PMAC for **COMMAND** statements.

If the control-panel inputs are disabled by I2=1, the host-addressed coordinate system also controls the indicator lines for the in-position, warning-following-error, and fatal-following-error functions. These indicator lines connect to both control-panel port outputs (all PMAC versions), and to the interrupt controller (PMAC-PC, PMAC-Lite, PMAC-STD). (If I2=0, the hardware-selected coordinate system controls these lines.)

Example

```
&1B4R ..... ; C.S.1 point to Beginning of Prog 4 and Run
Q ..... ; C.S.1 Quit running program
&3B6R ..... ; C.S.3 point to Beginning of Prog 5 and Run
A ..... ; C.S.3 Abort program
```

See Also I-variable I2
On-line commands **#**, **#{motor number}**, **&**
Program commands **ADDRESS**, **COMMAND**

&

Function Report currently addressed coordinate system.

Scope Global

Syntax **&**

Remarks This command causes PMAC to return the number of the coordinate system currently addressed by the host.

Note that a different coordinate system may be hardware selected from the control panel port for coordinate-system-specific control panel inputs, and that different coordinate systems may be addressed from programs within PMAC for **COMMAND** statements.

Example

```
&..... ; Ask PMAC which C.S. is addressed
4 ; PMAC reports that C.S. 4 is addressed
```

See Also I-variable I2
On-line commands **#, #{motor number},&{C.S. number}**;
Program commands **ADDRESS, COMMAND**;

< {Option 6L firmware only}

Function Back-up through Lookahead Buffer

Scope Coordinate-system specific

Syntax **<**

Remarks This command causes the PMAC to start reverse execution in the lookahead buffer for the addressed coordinate system. If the program is currently executing in the forward direction, it will be brought to a quick stop (the equivalent of the **** command) first.

Execution proceeds backward through points buffered in the lookahead buffer, observing velocity and acceleration constraints just as in the forward direction. This execution continues until one of the following occurs:

- Reverse execution reaches the beginning of the lookahead buffer – the oldest stored point still remaining in the lookahead buffer – and it comes to a controlled stop at this point, observing acceleration limits in decelerating to a stop.
- The **** quick-stop command is given, which causes PMAC to come to the quickest possible stop in the lookahead buffer.
- The **>** resume-forward, **R** run, or **S** step command is given, which causes PMAC to resume normal forward execution of the program, adding to the lookahead buffer as necessary.
- An error condition occurs, or a non-recoverable stopping command is given.

If any motor has been jogged away from the quick-stop point, and not returned with a **J=** command, PMAC will reject the **<** back-up command, reporting ERR017 if I6 is set to 1 or 3.

This same functionality can be obtained from within a PMAC program by setting I1021 to 7, which executes more quickly than **CMD &n<**.

If the coordinate system is not currently in the middle of a lookahead sequence, PMAC will treat this command as an **H** feed-hold command.

> {Option 6L firmware only}

Function Resume Forward Execution in Lookahead Buffer

Scope Coordinate-system specific

Syntax **>**

Remarks This command causes the PMAC to resume forward execution in the lookahead buffer for the addressed coordinate system. It is typically used to resume normal operation after a \ quick-stop command, or a **<** back-up command. If the program is currently executing in the backward direction, it will be brought to a quick stop (the equivalent of the **** command) first.

If previous forward execution had been in continuous mode (started with the **R** command), the **>** command will resume it in continuous mode. If previous forward execution had been in single-step mode (started with the **S** command), the **>** command will resume it in single-step mode. The R and S commands can also be used to resume forward execution, but they may change the continuous/single-step mode.

Deceleration from a backward move (if any) and acceleration in the forward direction observe the Ix17 acceleration limits.

If any motor has been jogged away from the quick-stop point, and not returned with a **J=** command, PMAC will reject the **>** resume command, reporting ERR017 if I6 is set to 1 or 3.

This same functionality can be obtained from within a PMAC program by setting I1021 to 6, which executes more quickly than **CMD &n>**.

If the coordinate system is not currently in the middle of a lookahead sequence, PMAC will treat this command as an **R** run command.

/

Function Halt program execution at end of currently executing move

Scope Coordinate-system specific

Syntax **/**

Remarks This command causes PMAC to halt the execution of the motion program running in the currently addressed coordinate system at the end of the currently executing move, provided PMAC is in segmentation mode (I13>0). If PMAC is not in segmentation mode (I13=0), the **/** command has the same effect as the **Q** command, halting execution at the end of the latest *calculated* move, which can be 1 or 2 moves past the currently *executing* move.

If the coordinate system is currently executing moves with the special lookahead function (Option 6L firmware only), motion will stop at the end of the move currently being *added* to the lookahead buffer. This is not necessarily the move that is currently executing from the lookahead buffer, and there can be a significant delay before motion is halted. Acceleration limits will be observed while ramping down to a stop at the programmed point.

Once halted at the end of the move, program execution can be resumed with the **R** command. In the meantime, the individual motors may be jogged way from this point, but they must all be returned to this point using the **J=** command before program execution may be resumed. An attempt to resume program execution from a different point will result in an error (ERR017 reported if I6 = 1 or 3). If resumption of this program from this point is not desired, the **A** (abort) command should be issued before other programs are run.

Example

```
&1B5R.....          ; Command C.S. 1 to start PROG 5
/.....              ; Halt execution of program
#1J+.....           ; Jog Motor 1 positive
J/ .....            ; Stop jogging
J= .....            ; Return to prejog position
R.....              ; Resume execution of PROG 5
/.....              ; Halt program execution
#2J-.....           ; Jog Motor 2 negative
J/ .....            ; Stop jogging
R.....              ; Try to resume execution of PROG 5
<BELL>ERR017        ; PMAC reports error; not at position to resume
J= .....            ; Return to prejog position
R                   ; Resume execution of PROG 5
```

See Also I-variables I6, I13
On-line commands **R**, **J=**, **Q**, **A**, ****, **H**

?

Function Report motor status

Scope Motor specific

Syntax **?**

Remarks This command causes PMAC to report the motor status bits as an ASCII hexadecimal word. PMAC returns twelve characters, representing two status words. Each character represents four status bits. The first character represents Bits 20-23 of the first word; the second shows Bits 16-19; and so on, to the sixth character representing Bits 0-3. The seventh character represents Bits 20-23 of the second word; the twelfth character represents Bits 0-3.

The value of a bit is 1 when the condition is true; 0 when it is false. The meaning of the individual bits is:

FIRST WORD RETURNED (X:$003D, X:$0079, etc.):

First character returned:

Bit 23 *Motor Activated*: This bit is 1 when Ix00 is 1 and the motor calculations are active; it is 0 when Ix00 is 0 and motor calculations are deactivated.

Bit 22 *Negative End Limit Set*: This bit is 1 when motor actual position is less than the software negative position limit (Ix14), or when the hardware limit on this end (+LIMn – note!) has been tripped; it is 0 otherwise. If the motor is deactivated (bit 23 of the first motor status word set to zero) or killed (bit 14 of the second motor status word set to zero) this bit is not updated.

Bit 21 *Positive End Limit Set*: This bit is 1 when motor actual position is greater than the software positive position limit (Ix13), or when the hardware limit on this end (-LIMn – note!) has been tripped; it is 0 otherwise. If the motor is deactivated (bit 23 of the first motor status word set to zero) or killed (bit 14 of the second motor status word set to zero) this bit is not updated.

Bit 20 *Handwheel Enabled*: This bit is 1 when Ix06 is 1 and position following for this axis is enabled; it is 0 when Ix06 is 0 and position following is disabled.

Second character returned:

Bit 19 *Phased Motor*: This bit is 1 when Ix01 is 1 and this motor is being commutated by PMAC; it is 0 when Ix01 is 0 and this motor is not being commutated by PMAC.

Bit 18 *Open Loop Mode*: This bit is 1 when the servo loop for the motor is open, either with outputs enabled or disabled (killed). (Refer to Amplifier Enabled status bit to distinguish between the two cases.) It is 0 when the servo loop is closed (under position control, always with outputs enabled).

Bit 17 *Running Definite-Time Move*: This bit is 1 when the motor is executing any move with a predefined end-point and end-time. This includes any motion program move dwell or delay, any jog-to-position move, and the portion of a homing search move after the trigger has been found. It is 0 otherwise. It changes from 1 to 0 when execution of the *commanded* move finishes.

Bit 16 *Integration Mode*: This bit is 1 when Ix34 is 1 and the servo loop integrator is only active when desired velocity is zero. It is 0 when Ix34 is 0 and the servo loop integrator is always active.

Third character returned:

Bit 15 *Dwell in Progress*: This bit is 1 when the motor's coordinate system is executing a **DWELL** instruction. It is 0 otherwise.

Bit 14 *Data Block Error*: This bit is 1 when move execution has been aborted because the data for the next move section was not ready in time. This is due to insufficient calculation time. It is 0 otherwise. It changes from 1 to 0 when another move sequence is started. This is related to the *Run Time Error* Coordinate System status bit.

Bit 13 *Desired Velocity Zero*: This bit is 1 if the motor is in closed-loop control and the commanded velocity is zero (i.e. it is trying to hold position). It is zero either if the motor is in closed-loop mode with non-zero commanded velocity, or if it is in open-loop mode.

Bit 12 *Abort Deceleration*: This bit is 1 if the motor is decelerating due to an Abort command, or due to hitting hardware or software position (overtravel) limits. It is 0 otherwise. It changes from 1 to 0 when the *commanded* deceleration to zero velocity finishes.

Fourth character returned:

Bit 11 *Block Request*: This bit is 1 when the motor has just entered a new move section, and is requesting that the upcoming section be calculated. It is 0 otherwise. It is primarily for internal use.

Bit 10 *Home Search in Progress*: This bit is set to 1 when the motor is in a move searching for a trigger: a homing search move, a jog-until trigger, or a motion program move-until-trigger. It becomes 1 as soon as the calculations for the move have started, and becomes zero again as soon as the trigger has been found, or if the move is stopped by some other means. This is *not* a good bit to observe to see if the full move is complete, because it will be 0 during the post-trigger portion of the move. Use the Home Complete and Desired Velocity Zero bits instead.

Bits 8-9 These bits are used to store a pointer to the next data block for motor calculations. They are primarily for internal use.

Fifth and sixth characters returned:

Bits 0-7 These bits are used to store a pointer to the next data block for motor calculations. They are primarily for internal use.

SECOND WORD RETURNED (Y:$0814, Y:$08D4, etc.):

Seventh character returned:

Bit 23 *Assigned to C.S.*: This bit is 1 when the motor has been assigned to an axis in any coordinate system through an axis definition statement. It is 0 when the motor is not assigned to an axis in any coordinate system.

Bits 20-22 *(C.S. - 1) Number*: These three bits together hold a value equal to the (Coordinate System number minus one) to which the motor is assigned. Bit 22 is the MSB, and bit 20 is the LSB. For instance, if the motor is assigned to an axis in C. S. 6, these bits would hold a value of 5: bit 22 =1, bit 21 = 0, and bit 20 = 1.

Eighth character returned:

Bits 16-19 (Reserved for future use)

Ninth Character Returned:

Bit 15 (Reserved for future use)

Bit 14 *Amplifier Enabled*: This bit is 1 when the outputs for this motor's amplifier are enabled, either in open-loop or closed-loop mode (refer to Open-Loop Mode status bit to distinguish between the two cases). It is 0 when the outputs are disabled (killed).

Bits 12-13 (Reserved for future use)

Tenth Character Returned:

Bit 11 *Stopped on Position Limit*: This bit is 1 if this motor has stopped because of either a software or a hardware position (overtravel) limit, *even if the condition that caused the stop has gone away*. It is 0 at all other times, even when into a limit but moving out of it.

Bit 10 *Home Complete*: This bit, set to 0 on power-up or reset, becomes 1 when the homing move *successfully* locates the home trigger. At this point in time the motor is usually decelerating to a stop or moving to an offset from the trigger determined by Ix26. If a second homing move is done, this bit is set to 0 at the beginning of the move, and only becomes 1 again if that homing move *successfully* locates the home trigger. Use the *Desired Velocity Zero* bit and/or the *In Position* bit to monitor for the end of motor motion.

Bit 9 (Reserved for future use)

Bit 8 *Phasing Search Error*: This bit is set to 1 if the phasing search move for a PMAC-commutated motor has failed due to amplifier fault, overtravel limit, or lack of detected motion. It is set to 0 if the phasing search move did not fail by any of these conditions (not an absolute guarantee of a successful phasing search).

Eleventh Character Returned:

Bit 7 *Trigger Move*: This bit is set to 1 at the beginning of a jog-until-trigger or motion program move-until-trigger. It is set to 0 at the end of the move if the trigger has been found, but remains at 1 if the move ends with no trigger found. This bit is useful to determine whether the move was successful in finding the trigger.

Bit 6 *Integrated Fatal Following Error*: This bit is 1 if this motor has been disabled due to an integrated following error fault, as set by Ix11 and Ix63. The fatal following error bit (bit 2) will also be set in this case. Bit 6 is zero at all other times, becoming 0 again when the motor is re-enabled.

Bit 5 *I^2T Amplifier Fault Error*: This bit is 1 if this motor has been disabled by an integrated current fault. The amplifier fault bit (bit 3) will also be set in this case. Bit 5 is 0 at all other times, becoming 0 again when the motor is re-enabled.

Bit 4 *Backlash Direction Flag*: This bit is 1 if backlash has been activated in the negative direction. It is 0 otherwise.

Twelfth Character Returned:

Bit 3 *Amplifier Fault Error*: This bit is 1 if this motor has been disabled because of an amplifier fault signal, *even if the amplifier fault signal has gone away*, or if this motor has been disabled due to an I^2T integrated current fault (in which case bit 5 is also set). It is 0 at all other times, becoming 0 again when the motor is re-enabled.

Bit 2 *Fatal Following Error*: This bit is 1 if this motor has been disabled because it exceeded its fatal following error limit (Ix11) or because it exceeded its integrated following error limit (Ix63; in which case bit 6 is also set). It is 0 at all other times, becoming 0 again when the motor is re-enabled.

Bit 1 *Warning Following Error*: This bit is 1 if the following error for the motor exceeds its warning following error limit (Ix12). It stays at 1 if the motor is killed due to fatal following error. It is 0 at all other times, changing from 1 to 0 when the motor's following error reduces to under the limit, or if killed, is re-enabled.

Bit 0 *In Position*: This bit is 1 when five conditions are satisfied: the loop is closed, the desired velocity zero bit is 1 (which requires closed-loop control and no commanded move); the program timer is off (not currently executing any move, **DWELL**, or **DELAY**), the magnitude of the following error is smaller than Ix28.and the first four conditions have been satisfied for (I7+1) consecutive scans.

Example

```
#1?.....                ; Request status of Motor 1
812000804401 ...        ; PMAC responds with 12 hex digits representing 48 bits
.....                   ; The following bits are true (all others are false)
.....                   ; Word 1 Bit 23: Motor Activated
.....                   ;   Bit 16: Integration Mode
.....                   ;   Bit 13: Desired Velocity Zero
.....                   ; Word 2 Bit 23: Assigned to Coordinate System
.....                   ;   (Bits 20-22 all 0 – assigned to C.S.1)
.....                   ;   Bit 14: Amplifier Enabled
.....                   ;   Bit 10: Home Complete
                        ;   Bit 0: In Position
```

See Also On-line commands **<CTRL-B>**, **??**, **???**
Memory map registers X:$003D, X:$0079, etc. Y:$0814, Y:$08D4, etc.
Suggested M-variable definitions Mx30-Mx45

??

Function Report the status words of the addressed coordinate system.

Scope Coordinate-system specific

Syntax **??**

Remarks This causes PMAC to report status bits of the addressed coordinate system as an ASCII hexadecimal word. PMAC returns twelve characters, representing two status words. Each character represents four status bits. The first character represents bits 20-23 of the first word; the second shows bits 16-19; and so on, to the sixth character representing bits 0-3. The seventh character represents bits 20-23 of the second word; the twelfth character represents its 0-3.

The value of a bit is 1 when the condition is true; 0 when it is false. The meanings of the individual bits are:

FIRST WORD RETURNED (X:$0818, X:$08D8, etc.)

First character returned:

Bit 23 *Z-Axis Used in Feedrate Calculations*: This bit is 1 if this axis is used in the vector feedrate calculations for F-based moves in the coordinate system; it is 0 if this axis is not used. See the **FRAX** command.

Bit 22 *Z-Axis Incremental Mode*: This bit is 1 if this axis is in incremental mode – moves specified by distance from the last programmed point. It is 0 if this axis is in absolute mode – moves specified by end position, not distance. See the **INC** and **ABS** commands.

Bit 21 *Y-Axis Used in Feedrate Calculations*: (See bit 23 description.)

Bit 20 *Y-Axis Incremental Mode*: (See bit 22 description.)

Second character returned:

Bit 19 *X-Axis Used in Feedrate Calculations*: (See bit 23 description.)

Bit 18 *X-Axis Incremental Mode*: (See bit 22 description.)

Bit 17 *W-Axis Used in Feedrate Calculations*: (See bit 23 description.)

Bit 16 *W-Axis Incremental Mode*: (See bit 22 description.)

Third character returned:

Bit 15 *V-Axis Used in Feedrate Calculations*: (See bit 23 description.)

Bit 14 *V-Axis Incremental Mode*: (See bit 22 description.)

Bit 13 *U-Axis Used in Feedrate Calculations*: (See bit 23 description.)

Bit 12 *U-Axis Incremental Mode*: (See bit 22 description.)

Fourth character returned:

Bit 11 *C-Axis Used in Feedrate Calculations*: (See bit 23 description.)

Bit 10 *C-Axis Incremental Mode*: (See bit 22 description.)

Bit 9 *B-Axis Used in Feedrate Calculations*: (See bit 23 description.)

Bit 8 *B-Axis Incremental Mode*: (See bit 22 description.)

Fifth character returned:

Bit 7 *A-Axis Used in Feedrate Calculations*: (See bit 23 description.)

Bit 6 *A-Axis Incremental Mode*: (See bit 22 description.)

Bit 5 *Radius Vector Incremental Mode*: This bit is 1 if circle move radius vectors are specified incrementally (i.e. from the move start point to the arc center). It is 0 if circle move radius vectors are specified absolutely (i.e. from the XYZ origin to the arc center). See the **INC(R)** and **ABS(R)** commands.

Bit 4 *Continuous Motion Request*: This bit is 1 if the coordinate system has requested of it a continuous set of moves (e.g. with an **R** command). It is 0 if this is not the case (e.g. not running program, Ix92=1, or running under an **S** command).

Sixth character returned:

Bit 3 *Move-Specified-by-Time Mode*: This bit is 1 if programmed moves in this coordinate system are currently specified by time (TM or TA), and the move speed is derived. It is 0 if programmed moves in this coordinate system are currently specified by feedrate (speed; F) and the move time is derived.

Bit 2 *Continuous Motion Mode*: This bit is 1 if the coordinate system is in a sequence of moves that it is blending together without stops in between. It is 0 if it is not currently in such a sequence, for whatever reason.

Bit 1 *Single-Step Mode*: This bit is 1 if the motion program currently executing in this coordinate system has been told to "step" one move or block of moves, or if it has been given a Q (Quit) command. It is 0 if the motion program is executing a program by an R (run) command, or if it is not executing a motion program at all.

Bit 0 *Running Program*: This bit is 1 if the coordinate system is currently executing a motion program. It is 0 if the C.S. is not currently executing a motion program. Note that it becomes 0 as soon as it has *calculated* the last move and reached the final **RETURN** statement in the program, even if the motors are still *executing* the last move or two

that have been calculated. Compare to the motor *Running Program* status bit.

SECOND WORD RETURNED (Y:$0817, Y:$08D7, etc.)

Seventh character returned:

Bit 23 *Program Hold Stop*: This bit is 1 when a motion program running in the currently addressed Coordinate System is stopped using the ‘ \ ‘ command from a segmented move (LINEAR or CIRCLE mode with I13 > 0).

Bit 22 *Run-Time Error*: This bit is 1 when the coordinate system has stopped a motion program due to an error encountered while executing the program (e.g. jump to non-existent label, insufficient calculation time, etc.)

Bit 21 *Circle Radius Error*: This bit is 1 when a motion program has been stopped because it was asked to do an arc move whose distance was more than twice the radius (by an amount greater than Ix96).

Bit 20 *Amplifier Fault Error*: This bit is 1 when any motor in the coordinate system has been killed due to receiving an amplifier fault signal. It is 0 at other times, changing from 1 to 0 when the offending motor is re-enabled.

Eighth character returned:

Bit 19 *Fatal Following Error*: This bit is 1 when any motor in the coordinate system has been killed due to exceeding its fatal following error limit (Ix11). It is 0 at other times. The change from 1 to 0 occurs when the offending motor is re-enabled.

Bit 18 *Warning Following Error*: This bit is 1 when any motor in the coordinate system has exceeded its warning following error limit (Ix12). It stays at 1 if a motor has been killed due to fatal following error limit. It is 0 at all other times. The change from 1 to 0 occurs when the offending motor’s following error is reduced to under the limit, or if killed on fatal following error as well, when it is re-enabled.

Bit 17 *In Position*: This bit is 1 when *all* motors in the coordinate system are “in position”. Five conditions must apply for all of these motors for this to be true:, the loops must be closed, the desired velocity must be zero for all motors, the coordinate system cannot be in any timed move (even zero distance) or DWELL, all motors must have a following error smaller than their respective Ix28 in-position bands, and the above conditions must have been satisfied for (I7+1) consecutive scans.

Bit 16 *Rotary Buffer Request*: This bit is 1 when a rotary buffer exists for the coordinate system and enough program lines have been sent to it so that the buffer contains at least I17 lines ahead of what has been calculated. Once this bit has been set to 1 it will not be set to 0 until there are less than I16 program lines ahead of what has been calculated. The ‘PR’ command may be used to find the current number of program lines ahead of what has been calculated.

Ninth character returned:

Bit 15 *Delayed Calculation Flag*: (for internal use)

Bit 14 *End of Block Stop*: This bit is 1 when a motion program running in the currently addressed Coordinate System is stopped using the ‘ / ‘ command from a segmented move (Linear or Circular mode with I13 > 0).

Bit 13 *Synchronous M-variable One-Shot*: (for internal use)

Bit 12 *Dwell Move Buffered*: (for internal use)

Tenth character returned:

Bit 11 *Cutter Comp Outside Corner*: This bit is 1 when the coordinate system is executing an added outside corner move with cutter compensation on. It is 0 otherwise.

Bit 10 *Cutter Comp Move Stop Request*: This bit is 1 when the coordinate system is executing moves with cutter compensation enabled, and has been asked to stop move execution. This is primarily for internal use.

Bit 9 *Cutter Comp Move Buffered*: This bit is 1 when the coordinate system is executing moves with cutter compensation enabled, and the next move has been calculated and buffered. This is primarily for internal use.

Bit 8 *Pre-jog Move Flag*: This bit is 1 when any motor in the coordinate system is executing a jog move to "pre-jog" position (**J=** command). It is 0 otherwise.

Eleventh character returned:

Bit 7 *Segmented Move in Progress*: This bit is 1 when the coordinate system is executing motion program moves in segmentation mode (I13>0). It is 0 otherwise. This is primarily for internal use.

Bit 6 *Segmented Move Acceleration*: This bit is 1 when the coordinate system is executing motion program moves in segmentation mode (I13>0) and accelerating from a stop. It is 0 otherwise. This is primarily for internal use.

Bit 5 *Segmented Move Stop Request*: This bit is 1 when the coordinate system is executing motion program move in segmentation mode (I13>0) and it is decelerating to a stop. It is 0 otherwise. This is primarily for internal use.

Bit 4 *PVT/SPLINE Move Mode*: This bit is 1 if this coordinate system is in either PVT move mode or SPLINE move mode. (If bit 0 of this word is 0, this means PVT mode; if bit 0 is 1, this means SPLINE mode.) This bit is 0 if the coordinate system is in a different move mode (LINEAR, CIRCLE, or RAPID). See the table below.

Twelfth character returned:

Bit 3 *Cutter Compensation Left*: This bit is 1 if the coordinate system has cutter compensation on, and the compensation is to the left when looking in the direction of motion. It is 0 if compensation is to the right, or if cutter compensation is off.

Bit 2 *Cutter Compensation On*: This bit is 1 if the coordinate system has cutter compensation on. It is 0 if cutter compensation if off.

Bit 1 *CCW Circle\Rapid Mode*: When bit 0 is 1 and bit 4 is 0, this bit is set to 0 if the coordinate system is in CIRCLE1 (clockwise arc) move mode and 1 if the coordinate system is in CIRCLE2 (counterclockwise arc) move mode. If both bits 0 and 4 are 0, this bit is set to 1 if the coordinate system is in RAPID move mode. Otherwise, this bit is 0. See the table below.

Bit 0 *CIRCLE/SPLINE Move Mode*: This bit is 1 if the coordinate system is in either CIRCLE or SPLINE move mode. (If bit 4 of this word is 0, this means CIRCLE mode; if bit 4 is 1, this means SPLINE mode.) This bit is 0 if the coordinate system is in a different move mode (LINEAR, PVT, or RAPID.). See the table below.

The states of bits 4, 1, and 0 in the different move modes are summarized in the following table:

Mode	Bit 4	Bit 1	Bit 0
Linear	0	0	0
Rapid	0	1	0
Spline	1	0	1
Circle1	0	0	1
Circle2	0	1	1
PVT	1	1	0

Example

```
??  .....                ; Request coordinate system status words
A8002A020010             ; PMAC responds; the following bits are true:
.....                    ; Word 1 Bit 23: Z-axis used in feedrate calcs
.....                    ;   Bit 21: Y-axis used in feedrate calcs
.....                    ;   Bit 19: X-axis used in feedrate calcs
.....                    ;   Bit 5: Radius vector incremental mode
.....                    ;   Bit 3: Move specified by time
.....                    ;   Bit 1: Single-step mode
.....                    ; Word 2 Bit 17: In-position
                         ;   Bit 4: PVT/Spline mode
```

See Also On-line commands **<CONTROL-C>**, **?**, **???**
Memory-map registers X:$0818, X:$08D8, etc.; Y:$0817, Y:$08D7, etc.
Suggested M-variable definitions Mx80-Mx90

???

Function Report global status words.

Scope Global

Syntax **???**

Remarks This command causes PMAC to return the global status bits in ASCII hexadecimal form. PMAC returns twelve characters, representing two status words. Each character represents four status bits. The first character represents Bits 20-23 of the first word, the second shows Bits 16-19; and so on, to the sixth character representing Bits 0-3. The seventh character represents Bits 20-23 of the second word; the twelfth character represents Bits 0-3 of the second word.

A bit has a value of 1 when the condition is true; 0 when false. The meaning of the individual status bits is:

FIRST WORD RETURNED (X:$0003):

First character returned:

Bit 23 *Real-Time Interrupt Active*: This bit is 1 if PMAC is currently executing a real-time interrupt task (PLC 0 or motion program move planning). It is 0 if PMAC is executing some other task. Note: Communications can only happen outside of the real-time interrupt, so polling this bit will always return a value of 0. This bit is for internal use.

Bit 22 *Real-Time Interrupt Re-entry*: This bit is 1 if a real-time interrupt task has

taken long enough so that it was still executing when the next real-time interrupt came (I8+1 servo cycles later). It stays at 1 until the card is reset, or until this bit is manually changed to 0. If motion program calculations cause this, it is not a serious problem. If PLC 0 causes this (no motion programs running), it could be serious.

Bit 21 *Servo Active*: This bit is 1 if PMAC is currently executing servo update operations. It is 0 if PMAC is executing other operations. Note that communications can only happen outside of the servo update, so polling this bit will always return a value of 0. This bit is for internal use.

Bit 20 *Servo Error*: This bit is 1 if PMAC could not properly complete its servo routines. This is a serious error condition. It is 0 if the servo operations have been completing properly.

Second character returned:

Bit 19 *Data Gathering Function On*: This bit is 1 when the data gathering function is active; it is 0 when the function is not active.

Bit 18 *Data Gather to Start on Servo*: This bit is 1 when the data gathering function is set up to start on the next servo cycle. It is 0 otherwise. It changes from 1 to 0 as soon as the gathering function actually starts.

Bit 17 *Data Gather to Start on Trigger*: This bit is 1 when the data gathering function is set up to start on the rising edge of Machine Input 2. It is 0 otherwise. It changes from 1 to 0 as soon as the gathering function actually starts.

Bit 16 (Reserved for future use)

Third character returned:

Bit 15 (Reserved for future use)

Bit 14 *Leadscrew Compensation On*: This bit is 1 if leadscrew compensation is currently active in PMAC. It is 0 if the compensation is not active

Bit13 *Any Memory Checksum Error*: This bit is 1 if a checksum error has been detected for either the PMAC firmware or the user program buffer space. Bit 12 of this word distinguishes between the two cases.

Bit12 *PROM Checksum Error*: This bit is 1 if a firmware checksum error has been detected in PMAC's memory. It is 0 if a user program checksum error has been detected, or if no memory checksum error has been detected. Bit 13 distinguishes between these two cases.

Fourth character returned:

Bit 11 *DPRAM Error*: This bit is 1 if PMAC has detected an error in DPRAM communications. It is 0 otherwise.

Bit 10 *EAROM Error*: This bit is 1 if PMAC detected a checksum error in reading saved data from the EAROM (in which case it replaces this with factory defaults). It is 0 otherwise.

Bits 8-9 (for internal use)

fifth character returned:

Bit 7 (for internal use)

Bit 6 *TWS Variable Parity Error*: This bit is 1 if the most recent TWS-format M-

variable read or write operation with a device supporting parity had a parity error; it is 0 if the operation with such a device had no parity error. The bit status is indeterminate if the operation was with a device that does not support parity.

Bit 5 *MACRO Auxiliary Communications Error*: This bit is 1 if the most recent MACRO auxiliary read or write command has failed. It is set to 0 at the beginning of each MACRO auxiliary read or write command.

Bit 4 *MACRO Ring Check Error:* This bit is 1 if the MACRO ring check function is enabled (I1001 > 0) and PMAC has either detected I1004 ring communication errors in an I1001 servo-cycle period, or has failed to detect the receipt of I1005 ring sync packet.

Sixth character returned:

Bits 2-3 (Reserved for future use)

Bit 1 *All Cards Addressed*: This bit is set to 1 if all cards on a serial daisychain have been addressed simultaneously with the **@@** command. It is 0 otherwise.

Bit 0 *This Card Addressed*: This bit is set to 1 if this card is on a serial daisychain and has been addressed with the **@n** command. It is 0 otherwise.

SECOND WORD RETURNED (Y:$0003)

Seventh character returned:

Bit 23 (For internal use)

Bit 22 *Host Communication Mode*: This bit is 1 when PMAC is prepared to send its communications over the "host port" (PC bus or STD bus). It is 0 when PMAC is prepared to send its communications over the VMEbus or the serial port. It changes from 0 to 1 when it receives an alphanumeric command over the host port. It changes from 1 to 0 when it receives a **<CTRL-Z>** over the serial port.

Bits 20-21 (For Internal Use)

Eighth character returned:

Bit 19 *Motion Buffer Open*: This bit is 1 if any motion program buffer (PROG or ROT) is open for entry. It is 0 if none of these buffers is open.

Bit 18 *Rotary Buffer Open*: This bit is 1 if the rotary motion program buffer(s) (ROT) is (are) open for entry. It is 0 if this is (these are) closed.

Bit 17 *PLC Buffer Open*: This bit is 1 if a PLC program buffer is open for entry. It is 0 if none of these buffers is open.

Bit 16 *PLC Command*: This bit is 1 if PMAC is processing a command issued from a PLC or motion program through a CMD" " statement. It is 0 otherwise. It is primarily for internal use.

Ninth character returned:

Bit 15 *VME Communication Mode*: This bit is 1 when PMAC is prepared to send its communications over the VME bus "mailbox" port. It is 0 when PMAC is prepared to send its communications over the "host port" (PC bus or STD bus) or the serial port. It changes from 0 to 1 when it receives an alphanumeric command over the VME bus mailbox port. It changes from 1 to 0 when it receives a **<CTRL-Z>** over the serial port.

Bits 12-14 (For Internal use)

Tenth character returned:

Bit 11 *Fixed Buffer Full*: This bit is 1 when no fixed motion (PROG) or PLC buffers are open, or when one is open but there are less than I18 words available. It is 0 when one of these buffers is open and there are more than I18 words available.

Bits 8-10 (Internal use)

Eleventh and twelfth characters returned:

Bits 0-7 (Reserved for future use)

Example

```
???.....................   ; Ask PMAC for global status words
003000400000              ; PMAC returns the global status words
.........................  ; 1st word bit 13 (Any checksum error) is true;
.........................  ; 1st word bit 12 (PROM checksum error) is true;
.........................  ; 2nd word bit 23 (for internal use) is true;
                           ; All other bits are false
```

See Also On-line commands **?**, **??**, **<CTRL-G>**
Memory registers X:$0003, Y:$0003.

@

Function Report currently addressed card on serial daisy-chain

Scope Global

Syntax **@**

Remarks This command causes the addressed PMAC on a serial daisy-chain to report its number to the host. If all cards are addressed, card @0 will return an **@** character.

I1 must be set to 2 or 3 for this command to be accepted. Otherwise, ERR003 is reported.

Example

```
@                 ;Ask PMAC chain which card is addressed
4                 ;PMAC @4 reports that it is addressed
```

See Also Addressing Commands (Talking to PMAC)
Multiple-Card Applications (Synchronizing PMAC to External Events)
I-variable I1
On-line commands **#**, **#{constant}**, **&**, **&{constant}**, **@{constant}**
Jumpers E40-E43 (PMAC-PC, -Lite, -VME)
Switches SW1-1 to SW1-4 (PMAC-STD)

@{card}

Function Address a card on the serial daisy-chain.

Scope Global

Syntax **@{card}**
where:

- **{card}** is a hexadecimal digit (0 to 9, A to F), representing the number of the card on the serial daisychain to be addressed; or the @ character, denoting that all cards are to be addressed simultaneously.

Remarks This command makes the PMAC board specified by **{card}** the addressed board on the serial daisychain. (the one on which subsequent commands will act). The addressing is modal, so all further commands will affect this board until a different board is addressed. At

power-up/reset, Board @0 is addressed.

I1 must be set to 2 or 3 for this command to be accepted. Otherwise, ERR003 is reported.

To address all cards simultaneously, use the **@@** command. Query commands (those requiring a data response) will be rejected in this mode.

This command should be used only when multiple PMAC cards are connected on a single serial cable. In this case, I-variable I1 should be set to 2 or 3 on all boards. A board's card number is selected by jumpers E40-E43 (PMAC-PC, -Lite, -VME) or switches SW1-1 to SW1-4 (PMAC-STD).

Note:

While not required, it is best to give a <CR> after an @{card} command before any other command, in order to give the formerly addressed card time to "tri-state" its outputs so as not to interfere with responses from the newly addressed card.

Example

```
I1=2@0.....          ; This sequence can be used the first time talking to multiple cards
.....                  on a chain to put them in the proper configuration
@0
#1J+.....            ; Jog motor 1 of Card 0.
@5
P20.....             ; Request the value of P20 on card @5
@@R                  ; All cards, addressed C.S. run active program
```

See Also

Addressing Commands (Talking to PMAC)
Multiple-Card Applications (Synchronizing PMAC to External Events)
I-variable I2
On-line commands **#**, **&**, **&{constant}**, **@**
Jumpers E40-E43 (PMAC-PC, -Lite, -VME)
Switches SW1-1 to SW1-4 (PMAC-STD)

\

Function Do a program hold (permitting jogging while in hold mode)

Scope Coordinate-system specific

Syntax \

Remarks This command causes PMAC to do a program hold of the currently addressed coordinate system in a manner that permits jogging of the motors in the coordinate system while in hold mode, provided PMAC is in a segmented move (**LINEAR** or **CIRCLE** mode with I13>0). If PMAC is in segmentation mode (I13=0, or other move mode), the **** command has the same effect as the **H** command, bringing the motors to a stop in the same way, but not permitting any moves while in feed hold mode.

The rate of deceleration to a stop in program hold mode, and from a stop on the subsequent **R** command, is controlled by I-variable I52. This global I-variable controls the rate for all coordinate systems.

Once halted in hold mode, program execution can be resumed with the **R** command. In the meantime, the individual motors may be jogged way from this point, but they must all be returned to this point using the **J=** command before program execution may be resumed. An attempt to resume program execution from a different point will result in an error (ERR017 reported if I6 = 1 or 3). If resumption of this program from this point is not desired, the **A** (abort) command should be issued before other programs are run.

If PMAC is executing moves inside the special lookahead buffer when this command is received (Option 6L firmware only), the rate of deceleration is the fastest that does not exceed the Ix17 acceleration limit or any motor. In lookahead mode, reversal along the path is also then possible with the < command.

Example

```
&1B5R.....          ; Command C.S. 1 to start PROG 5
\.....              ; Command feed hold of program
#1J+.....           ; Jog Motor 1 positive
J/ .....            ; Stop jogging (examine part here)
J= .....            ; Return to prejog position
R.....              ; Resume execution of PROG 5
\.....              ; Halt program execution
#2J-.....           ; Jog Motor 2 negative
J/ .....            ; Stop jogging
R.....              ; Try to resume execution of PROG 5
<BELL>ERR017        ; PMAC reports error; not at position to resume
J= .....            ; Return to prejog position
R                   ; Resume execution of PROG 5
```

See Also

Stop Commands (Making Your Application Safe)
I-variables I6, I13, I52, Ix95
On-line commands **R**, **J=**, **Q**, **A**, **/**, **H**

A

Function Abort all programs and moves in the currently addressed coordinate system.

Scope Coordinate-system specific

Syntax **A**

Remarks This command causes all axes defined in the current coordinate system to begin immediately to decelerate to a stop, aborting the currently running motion program (if any). It also brings any disabled (killed) or open-loop motors (defined in the current coordinate system) to an enabled zero-velocity closed-loop state.

If moving, each motor will decelerate its commanded profile at a rate defined by its own motor I-variable Ix15. If there is significant following error when the **A** command is issued, it may take a long time for the actual motion to stop. Although the command trajectory is brought to a stop at a definite rate, the actual position will continue to catch up to the commanded position for a longer time.

Note that a multi-axis system may not stay on its programmed path during this deceleration.

Note:

Abort commands are not meant to be recovered from gracefully. If you wish to resume easily, us the **H**, **Q**, **/**, or **** command instead.

Motion program execution may resume (if a motion program was in fact aborted) by issuing either an **R** or **S** command, but two factors must be considered. First, the starting positions for calculating the next move will be the original end positions of the aborted move unless the **PMATCH** command is issued or I14=1. Second, the move from the aborted position to the next move end position may not be possible or desirable. The **J=** command may be used to jog each motor in the coordinate system to the original end position of the aborted move, provided I13 is 0 (no segmentation mode).

Example

```
B1R.....            ; Start Motion Program 1
A.....              ; Abort the program
#1J=#2J=.....       ; Jog motors to original move-end position
R                   ; Resume program with next move
```

See Also Stop Commands (Making Your Application Safe)
Control-Panel Port STOP/ Input (Connecting PMAC to the Machine)
I-variables I13, I14, Ix15
On-line commands **<CONTROL-A>**, **H**, **J/**, **K**, **Q**
JPAN connector pin 10

ABS

Function Select absolute position mode for axes in addressed coordinate system.

Scope Coordinate-system specific

Syntax **ABS**
ABS ({axis}[,{axis}...])
where:

- **{axis}** is a letter (X, Y, Z, A, B, C, U, V, W) representing the axis to be specified, or the character R to specify radial vector mode

> ***Note:***
> No spaces are permitted in this command.

Remarks This command, without any arguments, causes all subsequent positions for all axes in the coordinate system in motion commands to be treated as absolute positions (this is the default condition). An **ABS** command with arguments causes the specified axes to be in absolute mode, and all others to remain unchanged.

If R is specified as one of the 'axes', the I, J, and K terms of the circular move radius vector specification will be specified in absolute form (i.e. as a vector from the origin, not from the move start point). An **ABS** command without any arguments does not affect this vector specification. The default radial vector specification is incremental.

If a motion program buffer is open when this command is sent to PMAC, the command will be entered into the buffer for later execution.

Example

```
ABS(X,Y) ..... ; X & Y made absolute – other axes and radial vector left unchanged
ABS ..... ; All axes made absolute – radial vector left unchanged
ABS(R) ; Radial vector made absolute – all axes left unchanged
```

See Also Circular Moves (Writing a Motion Program)
On-line command **INC**
Program commands **ABS**, **INC**

{axis}={constant}

Function Re-define the specified axis position.

Scope Coordinate-system specific

Syntax **{axis}={constant}**
where:

- **{axis}** is a letter from the set (X, Y, Z, U, V, W, A, B, C) specifying the axis whose present position is to be re-named;
- **{constant}** is a floating-point value representing the new name value for the axis' present position

Remarks This command re-defines the current axis position to be the value specified in **{constant}**, in user units (as defined by the scale factor in the axis definition). It can be used to relocate the origin of the coordinate system. This does not cause the specified axis to

move; it simply assigns a new value to the position..

Internally, a position bias register is written to which creates this new position offset. **PSET** is the equivalent motion program command.

Example

```
X=0 .....          ; Call axis X's current position zero
Z=5000             ; Re-define axis Z's position as 5000
```

See Also Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line command **Z**
Program commands **PSET**, **ADIS**, **IDIS**.
B{constant}

B

Function Point the addressed coordinate system to a motion program.

Scope Coordinate-system specific

Syntax **B{constant}**
where:

- **{constant}** is a floating point value from 0.0 to 32767.99999 representing the program and location to point the coordinate system to; with the integer part representing the program number, and the fractional part multiplied by 100,000 representing the line label (zero fractional part means the top of the program).

Remarks This command causes PMAC to set the program counter of the addressed coordinate system to the specified motion program and location. It is usually used to set the program counter to the *B*eginning of a motion program. The next R or S command will start execution at this point.

If **{constant}** is an integer, the program counter will point to the beginning of the program whose number matches **{constant}**. Fixed motion program buffers (PROG) can have numbers from 1 to 32,767. The rotary motion program carries program number 0 for the purpose of this command.

If **{constant}** is not an integer, the fractional part of the number represents the line label (**N** or **O**) in the program to which to point. The fractional value multiplied by 100,000 determines the number of the line label to which to point (it fills the fraction to 5 decimal places with zeros).

> ***Note:***
>
> If a motion program buffer (including ROTARY) is open when this command is sent to PMAC, the command is entered into the buffer for later execution, to be interpreted as a B-axis move command.

Example

```
B7 .....           ;points to the top of PROG 7
B0 .....           ;points to the top of the rotary buffer
B12.6 .....        ;points to label N60000 of PROG 12
B3.025R            ;points to label N2500 of PROG 3 and runs
```

See Also On-line commands **DEFINE ROT**,**R**, **S**
Program commands **B{data}**, **N{constant}**, **O{constant}**.

CHECKSUM

Function Report the firmware checksum value.

Scope Global

Syntax **CHECKSUM**
CHKS

Remarks This command causes PMAC to report the reference checksum value of the firmware revision that it is using. The value is reported as a hexadecimal ASCII string. This value was computed during the compilation of the firmware. It is mainly used for troubleshooting purposes.

The comparative checksum value that PMAC is continually computing by scanning the firmware in active memory is stored in X:$0794. As long as there is no checksum error, this comparative value is continually changing as PMAC continues its calculations. However, if during any pass of the checksum calculations, if the final comparative checksum value does not agree with the reference value, the calculations stop, and the final erroneous value is held in X:$0794.

Example

```
CHECKSUM          ; Request firmware reference checksum value
9FA263            ; PMAC returns hex value
```

See Also On-line commands **DATE**, **VERSION**

CLEAR

Function Erase currently opened buffer.

Scope Global (Coordinate-system specific for rotary motion program buffers)

Syntax **CLEAR**
CLR

Remarks This command empties the currently opened program, PLC, rotary, etc. buffer Typically, as you create a buffer file in your host computer, you will start with the **OPEN {buffer}** and **CLEAR** commands (even though these lines are technically not part of the buffer), and follow with your actual contents. This will allow you to easily edit buffers from your host and repeatedly download the buffers, erasing the old buffer's contents in the process.

Example

```
OPEN PROG 1.          ; Open motion program buffer 1
CLEAR.....            ; Clear out this buffer
F1000.....            ; Program really starts here!
X2500.....            ;...and ends on this line!
CLOSE.....            ; This closes the program buffer
OPEN PLC 3 CLEAR CLOSE ; This erases PLC 3
```

See Also Program Buffers (Talking to PMAC)
On-line commands **OPEN**, **CLOSE**, **DELETE**.

CLEARFAULT

Function Clear Geo PMAC fault display

Scope Global

Syntax **CLEARFAULT**
CLRF

Remarks This command clears the seven-segment fault display on the Geo PMAC controller/amplifier package. After this command is issued, the fault display will show a "0". However, if the fault-causing condition is still present, the fault display will immediately show that fault number again.

CLOSE

Function Close the currently opened buffer.

Scope Global

Syntax **CLOSE**
CLS

Remarks This closes the currently **OPEN**ed buffer. This should be used immediate after the entry of a motion, PLC, rotary, etc. buffer. If the buffer is left open, subsequent statements that are intended as on-line commands (e.g. **P1=0**) will get entered into the buffer instead. It is good practice to have **CLOSE** at the beginning and end of any file to be downloaded to PMAC.

When PMAC receives a **CLOSE** command, it automatically appends a **RETURN** statement to the end of the open program buffer.

If *any* PROGRAM or PLC in PMAC is improperly structured (e.g. no **ENDIF** or **ENDWHILE** to match an **IF** or **WHILE**), PMAC will report an ERR003 at the **CLOSE** command for any buffer until the problem is fixed.

Example

```
CLOSE ..............    ; This makes sure all buffers are closed
OPEN PROG 1.            ; Open motion program buffer 1
CLEAR ..............    ; Clear out this buffer
F1000 ..............    ; Program actually starts here!...
X2500 ......            ;...and ends on this line!
CLOSE ..............    ; This closes the program buffer
LIST PROG 1.            ; Request listing of closed program
F1000 ..............    ; PMAC starts listing
X2500 ..............
RETURN                  ; This was appended by the CLOSE command
```

See Also Program Buffers (Talking to PMAC)
On-line commands **OPEN**, **CLEAR**, **<CTRL-L>**, **<CTRL-U>**

{constant}

Function Assign value to variable P0, or to table entry.

Scope Global

Syntax **{constant}**
where:

- **{constant}** is a floating point value

Remarks This command is the equivalent of **P0={constant}**. That is, a value entered by itself on a command line will be assigned to P-variable P0. This allows simple operator entry of numeric values through a dumb terminal interface. Where the value goes is hidden from the operator; the PMAC user program must take P0 and use it as appropriate.

> ***Note:***
>
> If a special table on PMAC (e.g. **STIMULUS**, **COMP**) has been defined but not filled, a constant value will be entered into this table, not into P0.

Example In a motion program:

```
P0=-1 .....                          ; Set P0 to an "illegal" value
SEND"Enter number of parts in run:"
 .....                               ; Prompt operator at dumb terminal
 .....                               ; Operator simply needs to type in number
WHILE (P0<1) WAIT                    ; Hold until get legal response
P1=0.....                            ; Initialize part counter
WHILE (P0<P1)                        ; Loop once per part
   P1=P1+1
   ...
```

See Also On-line commands **OPEN COMP**, **OPEN STIMULUS**, **P{constant}={expression}**

DATE

Function Report PROM firmware revision date.

Scope Global

Syntax

```
DATE
DAT
```

Remarks This command causes PMAC to report the revision date of the PROM firmware revision it is using. The date is reported in the American style: mm/dd/yy (month/day/year).

Example

```
DATE                    ;Ask PMAC for firmware revision date
07/22/92                ;PMAC responds with July 22, 1992
```

See Also On-line command **VERSION**, **TYPE**

DEFINE BLCOMP

Function Define backlash compensation table

Scope Motor specific

Syntax

```
DEFINE BLCOMP {entries},{count length}
DEF BLCOMP {entries},{count length}
```

where:

- **{entries}** is a positive integer constant representing the number of values in the table;
- **{count length}** is a positive integer representing the span of the table in encoder counts of the motor.

Remarks This command establishes a backlash compensation table for the addressed motor. The next **{entries}** constants sent to PMAC will be placed into this table. The last item on the command line **{count length}** specifies the span of the backlash table in encoder counts of the motor. In use, if the motor position goes outside of the range 0 to count-length, the position is rolled over to within this range before the compensation is computed. The spacing between entries in the table is **{count length}** divided by **{entries}.**

On succeeding lines will be given the actual entries of the table as constants separated by spaces and or carriage return characters. The units of these entries are 1/16 count, and the entries must be integer values. The first entry is the correction at one spacing from the motor zero position (as determined by the most recent home search move or power-up/reset), the second entry is the correction two spacings away, and so on. The correction from the table at motor zero position is zero by definition.

The correction is the amount subtracted (added in the negative direction) from the nominal commanded position when the motor is moving in the negative direction to get the corrected position. The correction from the backlash table is added to the Ix86 constant backlash parameter before adjusting the motor position. Corrections from any leadscrew compensation tables that have this motor as the target motor are always active in both directions.

The last entry in the table represents the correction at **{count length}** distance from the motor's zero position. Since the table has the capability to roll over, this entry also represents the correction at the motor's zero position. For this reason, the last entry should virtually always be set to zero.

> ***Note:***
>
> PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any BLCOMP buffer exists for a lower numbered motor, or if any TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. BLCOMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

I51 must be set to 1 to enable the table.

See Also Backlash Compensation (Setting Up a Motor)
Leadscrew Compensation (Setting Up a Motor)
I-variables I99, Ix85, Ix86
On-line commands **DEFINE COMP**, **DELETE BLCOMP**

DEFINE COMP (one-dimensional)

Function Define Leadscrew Compensation Table

Scope Motor specific

Syntax **DEFINE COMP {entries}, [#{source}[D], [#{target},]] {count length}**
where:

- **{entries}** is a positive integer constant representing the number of numbers in the table;
- **{source}** (optional) is a constant from 1 to 8 representing the motor whose position controls which entries in the table are used for the active correction; if none is specified, PMAC assumes the source is the addressed motor; if a **D** is specified after the source motor number, the desired position of the motor is used to calculate the correction; otherwise the actual position is used;
- **{target}** (optional) is a constant from 1 to 8 representing the motor that receives the

correction; if none is specified, PMAC assumes the target is the addressed motor;

- **{count length}** is a positive integer representing the span of the table in encoder counts of the source motor.

Remarks This command establishes a leadscrew (position) compensation table assigned to the addressed motor. The next **{entries}** constants sent to PMAC will be placed into this table. Once defined, the tables are enabled and disabled with the variable I51.

The table "belongs" to the currently addressed motor, and unless otherwise specified in the command line, it will use the addressed motor both for source position data and as the target for its corrections. Each motor can only have one table that "belongs" to it (for a total of 8 tables in one PMAC), but it can act as a source or a target for multiple motors.

> ***Note:***
> PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any COMP buffer exists for a lower numbered motor, or if any TCOMP, BLCOMP, TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. COMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

It is possible to directly specify a source motor (with **#{source}**), or source and target motors (with **#{source},#{target}**), in this command. Either or both of them may be different from the motor to which the table belongs. (In other words, just because a table belongs to a motor does not necessarily mean that it affects or is affected by that motor's position.)

The table can operate as a function of either the desired (commanded) or actual position of the source motor. If a **D** is entered immediately after the source motor number (which must be explicitly declared here), the table operates as a function of the desired position of the source motor; if no **D** is entered, the table operates as a function of the actual position of the source motor.

The last item on the command line, **{count length}**, specifies the span of the compensation table in encoder counts of the source motor. In use, if the source motor position goes outside of the range 0 to count-length, the source position is "rolled over" to within this range before the correction is computed. The spacing between entries in the table is **{count length}** divided by **{entries}.**

On succeeding lines will be given the actual entries of the table. The units of these entries are 1/16 count, and the entries must be integer values. The first entry is the correction at one spacing from the motor zero position (as determined by the most recent home search move or power-up/reset), the second entry is the correction two spacings away, and so on. The correction is the amount *added* to the nominal position to get the corrected position. The correction at the zero position is zero by definition.

The last entry in the table represents the correction at **{count length}** distance from the source motor's zero position. Since the table has the capability to roll over, this entry also represents the correction at the source motor's zero position. For this reason, the last entry should virtually always be set to zero.

Example

```
#1 DEFINE COMP 4,2000                    ; Create table for motor 1
```

```
ERR003                                  ; PMAC rejects this command
DELETE GATHER                           ; Clear other buffers to allow loading
DELETE ROTARY
#8DEFINE COMP 500,20000                 ; Uses motor 8 actual position as source and
                                        ; motor 8 as target, ; 500 entries, spacing
                                        ; of 40 counts
#7DEFINE COMP 256,#3D,32768             ; Belongs to motor 7, uses motor 3 desired
                                        ; position as source, motor 7 as target, 256
                                        ; entries, spacing of 128 counts
#6 DEFINE COMP 400,#5,#4,40000          ; Belongs to motor 6, uses #5 as source, #4 as
                                        ; target, 400 entries, spacing of 100 counts
#5 DEFINE COMP 200,#1D,#1,30000         ; "Belongs" to motor 5, uses #1 desired position
                                        ; as source and target, 200 entries, spacing of
                                        ; 150 count
I51=1                                   ; Enable compensation tables
```

See Also Leadscrew compensation (Setting Up a Motor)
I-variable I51
On-line commands **{constant}**, **LIST COMP**, **LIST COMP DEF**, **DELETE COMP**, **DELETE GATHER**, **DELETE ROTARY**, **SIZE**

DEFINE COMP (two-dimensional)

Function Define two-dimensional leadscrew compensation table.

Scope Motor specific

Syntax

```
DEFINE COMP {Rows}.{Columns}, #{RowMotor}[D],
      [#{ColumnMotor}[D], [#{TargetMotor}]],
            {RowLength},{ColumnLength}
DEF COMP...
```

where:

- **{Rows}** is a positive integer constant representing the number of rows in the table, where each row represents a fixed location of the second (*column*) source motor;
- **{Columns}** is a positive integer constant representing the number of columns in the table, where each column represents a fixed location of the first (*row*) source motor;
- **{RowMotor}** is an integer constant from 1 to 8 representing the number of the first source motor; defaults to addressed motor; if a **D** is specified after the source motor number, the desired position of the motor is used to calculate the correction; otherwise the actual position is used;
- **{ColumnMotor}** is an integer constant from 1 to 8 representing the number of the second source motor; if a **D** is specified after the source motor number, the desired position of the motor is used to calculate the correction; otherwise the actual position is used;
- **{TargetMotor}** is an integer constant from 1 to 8 representing the number of the target motor; defaults to addressed motor;
- **{RowSpan}** is the span of the table, in counts, along the first (row) source motor's travel;
- **{ColumnSpan}** is the span of the table, in counts, along the second (column) source motor's travel.

Remarks This command establishes a two-dimensional position compensation table assigned to the addressed motor. The next *(Rows+1)*(Columns+1)-1* constants sent to PMAC will be placed into this table. This type of table is usually used to correct a motor position (X, Y, or Z-axis) as a function of the planar position of two motors (e.g. X and Y axes). Once defined, the tables are enabled and disabled with the variable I51.

The table belongs to the currently addressed motor, and unless otherwise specified in the command line, it will use the addressed motor both as the first-motor source position data and as the target for its corrections. Each motor can only have one table that belongs to it (for a total of eight tables in one PMAC), but it can act as a source and/or a target for multiple tables.

> ***Note:***
>
> PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any COMP buffer exists for a lower numbered motor, or if any TCOMP, BLCOMP, TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. COMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

The first source motor must be specified in the command line with **#{RowMotor}**. The second source motor may be specified in the command line with **#{ColumnMotor}**; if it is not specified, PMAC assumes that the second source motor is the currently addressed motor.

The target motor may be specified with **#{TargetMotor}**; if it is not specified, PMAC assumes that the target motor is the currently addressed motor.

In other words, if only one motor is specified in the command line, it is the first (row) source motor, and the second (column) source and target motors default to the addressed motor. If two motors are specified in the command line, the first one specified is the first (row) source motor, the second is the second (column) source motor, and the target motor defaults to the addressed motor. If three motors are specified, the first is the first (row) source motor, the second is the second (column) source motor, and the third is the target motor. None of these motors is required to be the addressed motor.

It is strongly recommended that you explicitly specify both source motors and the target motor in this command, to prevent possible confusion.

The table can operate as a function of either the desired (commanded) or actual position of the source motors. If a **D** is entered immediately after the source motor number (which must be explicitly declared here), the table operates as a function of the desired position of the source motor; if no **D** is entered, the table operates as a function of the actual position of the source motor. If the target motor is also one of the source motors, it is recommended that desired position be used, especially in high-gain systems, to prevent interaction with the servo dynamics.

The last two items on the command line, **{RowSpan}** and **{ColumnSpan}**, specify the span of the compensation table for the two source motors, row and column respectively, expressed in encoder counts of those motors. In use, if the source motor position goes outside of the range 0 to **{Span}**, the source position is "rolled over" to within this range along this axis before the correction is computed.

The count spacing between columns in the table is **{RowSpan}** divided by **{Columns}**. The count spacing between rows in the table is **{ColumnSpan}** divided by **{Rows}**. Note carefully the interaction between the row parameters and the column parameters.

On succeeding command lines will be given the actual correction entries of the table, given

as integer numerical constants in text form. The units of these entries are 1/16 count, and the entries must be integer values. The first entry is the correction at one column spacing from the zero position of the **RowMotor**, and the zero position of the **ColumnMotor**. The second entry is the correction at two column spacings from the zero position of the **RowMotor**, and the zero position of the **ColumnMotor**, and so on. Entry number **Columns** is the correction at **RowSpan** counts of the **RowMotor**, and at the zero position of the **ColumnMotor** (this entry should be zero if you wish to use the table along the edge, to match the implied zero correction at the origin). These entries should be considered as constituting "Row 0" of the table.

The next entry (entry **Columns**+1, the first entry of Row 1) is the correction at the zero position of the **RowMotor**, and one row spacing of the **ColumnMotor**. The following entry is the correction at one column spacing of the **RowMotor** and one row spacing of the **ColumnMotor**. The entry after this is the correction at two column spacing of the **RowMotor** and one row spacings of the **ColumnMotor**., and so on. The last entry of Row 1 (entry 2***Columns**+1) is the correction at one row spacing of the **RowMotor**, and **RowSpan** counts of the **ColumnMotor**.

Subsequent rows are added in this fashion, with the corrections of the entries for Row *n* being at *n* row spacings from the zero position of the **ColumnMotor**. The last row (row **Rows**) contains corrections at **ColumnSpan** counts of the **ColumnMotor**.

The size of the table is limited only by available data buffer space in PMAC's memory.

The following chart shows the order of entries into a 2D table with *r* rows and *c* columns, covering a span along the row motor of RowSpan, and along the column motor of ColSpan:

	Column Motor Position v	**Col 0**	**Col 1**	**Col 2**	**(Col j)**	**Col *c***
Row Motor Position >		0	RowSpan / c	2*RowSpan / c		RowSpan
Row 0	0	[0]	E_1	E_2	…	E_c
Row 1	ColSpan / r	E_{c+1}	E_{c+2}	E_{c+3}	…	E_{2c+1}
Row 2	2*ColSpan / r	E_{2c+2}	E_{2c+3}	E_{2c+4}	…	E_{3c+2}
(Row i)		…	…	…	(E_{ic+I+j})	…
Row *r*	ColSpan	E_{rc+r}	E_{rc+r+1}	E_{rc+r+2}		E_{rc+r+c}

There are several important details to note in the entry of a 2D table:

- The number of rows and number of columns is separated by a period, not a comma.
- The correction to the target motor at the zero position of both source motors is zero by definition. This is an implied entry at the beginning of the table (shown by [0] in the above chart); it should not be explicitly entered.
- Consecutive entries in the table are in the same row (except at row's end) separated by one column spacing of the position of the first source (row) motor.
- Both Row 0 and Row *r* must be entered into the table, so effectively you are entering (*r*+1) rows. If there is any possibility that you may go beyond an edge of the table, matching entries of Row 0 and Row *r* should have the same value to prevent a discontinuity in the correction. Row *r* in the table may simply be an added row beyond your real range of concern used just to prevent possible discontinuities at the edges of your real range of concern.
- Both Column 0 and Column *c* must be entered into the table, so effectively you are

entering (*c*+1) columns. If there is any possibility that you may go beyond an edge of the table, matching entries of Column 0 and Column *c* should have the same value to prevent a discontinuity in the correction. Column *c* in the table may simply be an added column beyond your real range of concern used just to prevent possible discontinuities at the edges of your real range of concern.

- Because the outside rows and outside columns must match each other to prevent edge discontinuities, the three explicitly entered corner corrections must be zero to match the implicit zero correction at the first corner of the table.

Example

```
#1 DEFINE COMP 40.30,#1,#2,#3,300000,400000
.....                          ; Create table belonging to Motor 1
ERR007.....                    ; PMAC rejects this command
DELETE GATHER                  ; Clear other buffers to allow loading
&1 DELETE ROTARY
&2 DELETE ROTARY
#2 DELETE COMP
#3 DELETE COMP
#4 DEFINE COMP 30.40,#1,#2,#3,400000,300000
.....                          ; Create same table, now belonging to Motor 4;
.....                          ; #1 & #2 are sources, #3 is target;
.....                          ; 30 rows x 40 columns, spacing of 10,000 counts
(1270 entries).....            ; (30+1)*(40+1)-1 entries of constants
#3 DEFINE COMP 25.20,#2,#3,#1,200000,250000
.....                          ; Create table belonging to Motor 3;
.....                          ; #2 and #3 are sources, #1 is target;
.....                          ; 25 rows x 20 columns, spacing of 10,000 counts
(545 entries).....             ;(25+1)*(20+1)-1 entries of constants
#2 DEFINE COMP 10.10,#1,#4,10000,20000
.....                          ; Create table belonging to Motor 2; #1 and #4 are
.....                          ; sources, #2 (default) is target; 10 rows x10 columns,
.....                          ; spacing of 1000 cts between  columns; pacing of
.....                          ; 2000 cts between rows
(120 entries).....             ; (10+1)*(10+1)-1 entries of constants
#1 DEFINE COMP 12.10,#4,1280,1200
.....                          ; Create table belonging to Motor 1, #4 and #1 (default)
.....                          ; are sources, #1 (default) is target 12 rows x 10 columns
.....                          ; spacing of 128 cts between columns spacing of 100 cts
.....                          ; between rows
(142 entries).....             ; (12+1)*(10+1)-1 entries of constants
I51=1 ......                    Enable compensation tables
```

See Also

Leadscrew compensation (Setting Up a Motor)
I-variable I51
On-line commands **{constant}**, **LIST COMP**, **LIST COMP DEF**, **DELETE COMP**, **DELETE GATHER**, **DELETE ROTARY**, **SIZE**

DEFINE GATHER

Function Create a data gathering buffer.

Scope Global

Syntax

```
DEFINE GATHER [{constant}]
DEF GAT [{constant}]
```

where:

- **{constant}** is a positive integer representing the number of words of memory to be reserved for the buffer

Remarks This command reserves space in PMAC's memory or in DPRAM depending upon the setting of I45 for the data gathering buffer and prepares it for collecting data at the beginning of the buffer. If a data gathering buffer already exists, its contents are not erased but the Data Gather Buffer Storage address (`Y:$0F20`) is reinitialized to the Data Gather Buffer Start address (`X:$0F20`) and the **LIST GATHER** command will no longer function. Data collection will again start at the beginning of the buffer when the command **GATHER** is issued.

If Data Gathering is in progress (an **ENDGATHER** command has not been issued and the gather buffer has not been filled up) PMAC will report an error on receipt of this command.

The optional **{constant}** specifies the number of long words to be reserved for the data gathering buffer, leaving the remainder of PMAC's memory available for other buffers such as motion and PLC programs. If **{constant}** is not specified, all of PMAC's unused buffer memory is reserved for data gathering. Until this buffer is deleted (with the **DELETE GATHER** command), no new motion or PLC programs may be entered into PMAC.

> ***Note:***
>
> If I45 = 2 or 3 the **{size}** requested in the **DEFINE GATHER {size}** command refers to a DPRAM long word (32-bits). If the **{size}** is smaller than required to hold an even multiple of the requested data, the actual data storage will go beyond the requested **{size}** to the next higher multiple of memory words required to hold the data. For example, if you are gathering one 24-bit value and one 48-bit value you will need 3 DPRAM long words of memory. If the **{size}** you specify is 4000 words (not a multiple of 3), the actual storage size will be 4002 words (the next higher multiple of 3).

The number of long words of unused buffer memory can be found by issuing the **SIZE** command.

Example

```
DEFINE GATHER
DEFINE GATHER 1000
```

See Also I-variables I19-I45
On-line commands **GATHER**, **DELETE GATHER**, **<CTRL-G>**, **SIZE**

DEFINE LOOKAHEAD {Option 6L firmware only}

Function Create lookahead buffer

Scope Coordinate-system specific

Syntax

```
DEFINE LOOKAHEAD {constant},{constant}
DEF LOOK {constant},{constant}
```

where:

- the first **{constant}** is a positive integer representing the number of move segments that can be stored in the buffer;
- the second **{constant}** is a positive integer representing the number of synchronous

M-variable assignments that can be stored in the buffer

Remarks This command establishes a lookahead buffer for the addressed coordinate system. It reserves memory to buffer both motion equations and "synchronous M-variable" output commands for the lookahead function.

PMAC can only have one lookahead buffer at a time. The coordinate system that is addressed when the lookahead buffer is defined is the only coordinate system that can execute the special lookahead function.

Segment Buffer Size: The first constant value in the command determines the number of motion segments that can be stored in the lookahead buffer. Each motion segment takes I13 milliseconds at the motion program speeds and acceleration times (the velocity and acceleration limits may extend these times).

However, it is the variable I1020 for the coordinate system that determines how many motion segments the coordinate system will actually look ahead in operation.

The lookahead buffer should be sized large enough to store all of the lookahead segments calculated, which means that this constant value must be greater than or equal to I1020. If backup capability is desired, the buffer must be sized to be large enough to contain the desired lookahead distance plus the desired backup distance.

The method for calculating the number of segments that must be stored ahead is explained in the I1020 specification and in the PMAC User's Guide section on Lookahead. The fundamental equation is:

$$I1020 = \frac{4}{3} * max\left[\frac{Ix16}{Ix17}\right]_x * \frac{1}{2 * I13}$$

If backup capability is desired, the buffer must be able to store an additional number of segments for the entire distance to be covered in reversal. This number of segments can be calculated as:

$$BackSegments = \frac{BackupDist(units) * 1000(m\,sec/\,sec)}{V_{max}(units/\,sec) * SegTime(m\,sec/\,seg)}$$

The total number of segments to reserve for the buffer is simply the sum of the forward and back segments you wish to be able to hold:

$$TotalSegments = I1020 + BackSegments$$

Memory Requirements: For each segment PMAC is told to reserve space for in the lookahead buffer, PMAC will reserve (*2x+4*) 48-bit words of memory from the main buffer memory space, where *x* is the number of motors in the coordinate system at the time that the buffer is defined. For example, if there are 5 motors in the coordinate system, a command to reserve space for 50 segments will reserve 50*(2*5 + 4) = 700 words of memory.

Once a lookahead buffer has been defined for a coordinate system, motors cannot be added to, or removed from, the coordinate system without upsetting the structure of the lookahead buffer. Attempting to do this will result in a "run-time" error on the next lookahead move.

If it is desired to add a motor to the coordinate system, or remove one, the lookahead buffer must first be erased with the **DELETE LOOKAHEAD** command, then re-defined after the change to the coordinate system has been made.

Output Buffer Size: The second constant value in the command determines the number of

synchronous M-variable assignments that can be stored in the lookahead buffer. Because these are evaluated at lookahead time, but not actually implemented until move execution time, they must be buffered. This section of the buffer must be large enough to store all of the assignments that could be made in the lookahead distance. Synchronous M-variable assignments are not made during backup, so there is no need to reserve memory to store assignments for the backup distance as well as the lookahead distance.

Memory Requirements: For each synchronous M-variable assignment PMAC is told to reserve space for in the lookahead buffer, PMAC will reserve two 48-bit words of memory.

There are no performance penalties for making the lookahead buffer larger than required, but this does limit the amount of PMAC memory free for other features.

A lookahead buffer is never retained through a power-down or board reset, so this command must be issued after every power-up/reset if the lookahead function is to be used.

To erase a lookahead buffer and free up the memory for other use, issue a **DELETE LOOKAHEAD** command, or reset the card.

PMAC will reject the **DEFINE LOOKAHEAD** command, reporting an *ERR003* if I6 = 1 or 3, if any lookahead buffer exists, or if a **GATHER** buffer exists. Any existing lookahead buffers and gather buffers must be deleted before a lookahead buffer can be defined.

DEFINE ROTARY

Function Define a rotary motion program buffer

Scope Coordinate-system specific

Syntax
```
DEFINE ROTARY{constant}
DEF ROT{constant}
```
where:

- **{constant}** is a positive integer representing the number of long words of memory to be reserved for the buffer

Remarks This command causes PMAC to create a rotary motion program buffer for the addressed coordinate system, allocating the specified number of long words of memory. Rotary buffers permit the downloading of program lines during the execution of the program.

The buffer should be large enough to allow it to hold safely the number of lines you anticipate downloading to PMAC ahead of the executing point. Each long word of memory can hold one sub-block of a motion program (i.e. **X1000 Y1000** is considered as two sub-blocks). The allocated memory for this coordinate system's rotary buffer remains resident until the buffer is deleted with **DELETE ROT**.

> ***Note:***
>
> PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any ROTARY buffer exists for a lower numbered coordinate system, or if a GATHER buffer exists. Any of these buffers must be deleted first. ROTARY buffers must be defined from high-numbered coordinate system to low-numbered coordinate system and deleted from low-numbered coordinate system to high-numbered coordinate system.

Example
```
DELETE GATHER                    ; Ensure open memory
```

```
&2DEFINE ROT 100                ; Create buffer for C.S. 2
&1DEFINE ROT 100                ; Create buffer for C.S. 1
&1B0 &2B0.....                  ; Point to these buffers
OPEN ROT.....                   ; Open these buffers for entry
  ... .....                     ; enter program lines here
```

See Also Rotary Program Buffers (Writing a Motion Program)
On-line commands **OPEN ROTARY**, **DELETE ROTARY**, **DELETE GATHER**

DEFINE TBUF

Function Create a buffer for axis transformation matrices.

Scope Global

Syntax
```
DEFINE TBUF {constant}
DEF TBUF {constant}
```
where:

- **{constant}** is a positive integer representing the number of transformation matrices to create

Remarks This command reserves space in PMAC's memory for one or more axis transformation matrices. These matrices can be used for real-time translation, rotation, scaling, and mirroring of the X, Y, and Z axes of any coordinate system. A coordinate system selects which matrix to use with the **TSELn** command, where n is an integer from 1 to the number of matrices created here.

> ***Note:***
>
> PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any ROTARYor GATHER buffer exists. Any of these buffers must be deleted first.

The number of long words of unused buffer memory can be found by issuing the **SIZE** command. Each defined matrix takes 21 words of memory.

Example
```
DELETE GATHER
DEF TBUF 1
DEFINE TBUF 8
```

See Also Axis Transformation Matrices (Writing a Motion Program)
On-line commands **DELETE TBUF**, **DELETE GATHER**, **SIZE**.
Program commands **TSEL**, **ADIS**, **AROT**, **IDIS**, **IROT**, **TINIT**

DEFINE TCOMP

Function Define torque compensation table

Scope Motor specific

Syntax
```
DEFINE TCOMP {entries},{count length}
DEF TCOMP {entries},{count length}
```
where:

- **{entries}** is a positive integer constant representing the number of values in the table;
- **{count length}** is a positive integer representing the span of the table in encoder counts of the motor.

Remarks This command establishes a torque compensation table for the addressed motor. The next **{entries}** constants sent to PMAC will be placed into this table. The last item on the command line **{count length}** specifies the span of the torque compensation table in encoder counts of the motor. In use, if the motor position goes outside of the range 0 to count-length, the position is "rolled over" to within this range before the compensation is computed. The spacing between entries in the table is **{count length}** divided by **{entries}.**

On succeeding lines will be given the actual entries of the table as constants separated by spaces and or carriage return characters. The entries are signed 24-bit integer values, with a range of –8,388,608 to +8,388,607. The full range of these entries corresponds to the full range of the 16-bit torque output of the servo loop, -32,768, to +32,767.

Therefore, an entry in the torque compensation table is numerically 256 times bigger than the corresponding servo-loop torque output.

The first entry is the correction at one spacing from the motor zero position (as determined by the most recent home search move or power-up/reset), the second entry is the correction two spacings away, and so on. PMAC computes corrections for positions between the table entries by a first-order interpolation between adjacent entries. The correction from the table at motor zero position is zero by definition.

The correction is the magnitude added to PMAC's servo loop output at that position. If PMAC's command is positive, a positive value from the table will increase the magnitude of the output; a negative value will decrease the magnitude of the output. If PMAC's command is negative, a positive value from the table will decrease the magnitude of the output in the negative direction; a negative value will increase the magnitude of the output.

The last entry in the table represents the correction at **{count length}** distance from the motor's zero position. Since the table has the capability to roll over, this entry also represents the correction at the motor's zero position. For this reason, the last entry should virtually always be set to zero.

Note:

PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any TCOMP buffer exists for a lower numbered motor, or if any BLCOMP, TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. TCOMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

I51 must be set to 1 to enable the table.

See Also Torque Compensation (Setting Up a Motor)
I-variables I51
On-line command **DELETE TCOMP**

DEFINE UBUFFER

Function Create a buffer for user variable use.

Scope Global

Syntax
```
DEFINE UBUFFER {constant}
DEF UBUF {constant}
```
where:

- **{constant}** is a positive integer representing the number of 48-bit words of PMAC memory to reserve for this purpose

Remarks This command reserves space in PMAC's memory for the user's discretionary use. This memory space will be untouched by any PMAC automatic functions. User access to this buffer is through M-variables, or possibly through on-line **W** (write) and **R** (read) commands.

The buffer starts at PMAC memory address $9FFF and continues back toward the beginning of memory ($0000) for the number of long (48-bit) words specified by {constant}. This memory space can be subdivided any way the user sees fit. On PMACs with battery backup, the values in the buffer at power-down will still be there at power-up. On PMACs with flash backup, the values in the buffer at the last **SAVE** command will be copied from the flash memory into the buffer at power-up or reset.

All other buffers except for fixed motion programs (PROG) and PLC programs must be deleted before PMAC will accept this command. There can be no rotary motion program, leadscrew compensation table, transformation matrix, data gathering or stimulus buffers in PMAC memory (any buffer created with a **DEFINE** command) for this command to be accepted. It is usually best to reinitialize the card with a **$$$***** command before sending the **DEFINE UBUFFER** command.

The address of the end of unreserved memory is held in register X:$0F3F. This register must hold the address $A000, signifying no defined buffers, in order for PMAC to be able to create a user buffer. Immediately after the user buffer has successfully been defined, this register will hold the address of the start of the buffer (the end of the user buffer is always at $9FFF). However, after other buffers have been defined, the end of unreserved memory will not match the beginning of the user buffer.

To free up this memory for other uses, the **DEFINE UBUFFER 0** command should be used (there is no **DELETE UBUFFER** command). It may be more convenient simply to re-initialize the board with a **$$$***** command.

Example

```
RHX:$0F3F                ; Look for end of unreserved memory
008A3D                   ; Reply indicates some reserved
$$$***                   ; Re-initialize card to clear memory
RHX:$0F3F                ; Check end of unreserved memory
00A000                   ; Reply indicates none reserved
DEFINE UBUFFER 256       ; Reserve memory for buffer
RHX:$0F3F                ; Check for beginning of buffer
009F00                   ; Reply confirms 256 words reserved
M1000->D:$9F00           ; Define M-variable to first word
M1010->Y:$9F80,12,1      ; Define M- variable to a middle word
M1023->X:$9FFF,24,S      ; Define M- variable to last word
```

See Also User Buffer, M-Variables (Computational Features)
On-line commands **$$$***, **R[H]{address}**, **W{address}**

DELETE BLCOMP

Function Erase backlash compensation table

Scope Motor specific

Syntax

```
DELETE BLCOMP
DEL BLCOMP
```

Remarks This command causes PMAC to erase the compensation table for the addressed motor, freeing that memory for other use.

> ***Note:***
>
> PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any BLCOMP buffer exists for a lower numbered motor, or if any TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be DELETEd first. BLCOMP buffers must be DEFINEd from high-numbered motor to low-numbered motor, and DELETEd from low-numbered motor to high-numbered motor.

Example

```
#2 DEL BLCOMP            ; Erase table belonging to Motor 2
ERR003.....              ; PMAC rejects this command
#1 DEL BLCOMP            ; Erase table belonging to Motor 1
#2 DEL BLCOMP            ; Erase table belonging to Motor 2
```

See Also Backlash Compensation (Setting Up a Motor)
I-variables I99, Ix85, Ix86
On-line command **DEFINE BLCOMP**

DELETE COMP

Function Erase leadscrew compensation table

Scope Motor specific

Syntax **DELETE COMP**
DEL COMP

Remarks This command causes PMAC to erase the compensation table belonging to the addressed motor, freeing that memory for other use.

> ***Note:***
>
> PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any COMP buffer exists for a lower numbered motor, or if any TCOMP, BLCOMP, TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. COMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

Remember that a compensation table *belonging* to a motor does not necessarily affect that motor or is not necessarily affected by that motor. The command **LIST COMP DEF** will tell which motors it affects and is affected by.

Example

```
#2DEL COMP...          ; Erase table belonging to Motor 2
ERR003.....            ; PMAC rejects this command
#1 DELETE COMP         ; Erase table belonging to Motor 1
#2 DELETE COMP         ; Erase table belonging to Motor 2
```

See Also Leadscrew compensation (Setting Up a Motor)
I-variable I51
On-line commands **{constant}**, **LIST COMP**, **LIST COMP DEF**, **DEFINE COMP**

DELETE GATHER

Function Erase the data gather buffer.

Scope Global

Syntax **DELETE GATHER**
DEL GAT

Remarks This command causes the data gathering buffer to be erased. The memory that was reserved is now de-allocated and is available for other buffers (motion programs, PLC programs, compensation tables, etc.). If data gathering is in progress (an **ENDGATHER** command has not been issued and the gather buffer has not been filled up) PMAC will report an error on receipt of this command.

PMAC's Executive Program automatically inserts this command at the top of a file when it uploads a buffer from PMAC into its editor, so the next download will not be hampered by an existing gather buffer. It is strongly recommended that you use this command as well when you create a program file in the editor (see Examples, below).

> ***Note:***
>
> When the executive program's data gathering function operates, it automatically reserves the entire open buffer space for gathered data. When this has happened, no additional programs or program lines may be entered into PMAC's buffer space until the **DELETE GATHER** command has freed this memory.

Example

```
CLOSE ..............    ; Make sure no buffers are open
DELETE GATHER          ; Free memory
OPEN PROG 50           ; Open new buffer for entry
CLEAR ..............   ; Erase contents of buffer
...                    ; Enter new contents here
```

See Also Buffered Commands (Talking to PMAC)
On-line commands **GATHER**, **DEFINE GATHER**, **SIZE**

DELETE LOOKAHEAD {Option 6L firmware only}

Function Erase lookahead buffer

Scope Coordinate-system specific

Syntax:
DELETE LOOKAHEAD
DEL LOOK

Remarks This command erases the lookahead buffer for the addressed coordinate system, freeing the memory for other uses. It also permits a motor to be added to, or removed from, the coordinate system, which cannot be done while there is a defined lookahead buffer for the coordinate system.

PMAC will reject the **DELETE LOOKAHEAD** command, reporting an *ERR003* if I6 = 1 or 3, if a data gathering buffer exists. An existing data gathering buffer must be erased with a **DELETE GATHER** command before the lookahead buffer may be deleted.

DELETE PLCC

Function Erase specified compiled PLC program

Scope Global

Syntax
DELETE PLCC {constant}
DEL PLCC {constant}
where:

- **{constant}** is an integer from 0 to 31, representing the program number

Remarks This command causes PMAC to erase the specified compiled PLC program. Remember that because all of the compiled PLC programs must be downloaded to PMAC together, the only way to restore this PLC is to download the entire set of compiled PLCs.

Only one PLCC program can be deleted in one command. Ranges and lists of PLCC program numbers are not permitted in this command.

To perform the same function for an uncompiled PLC program, the command sequence would be **OPEN PLC n CLEAR CLOSE**.

Example

```
DELETE PLCC 5          ; Erase compiled PLC program 5
DEL PLCC 0             ; Erase compiled PLC program 0
```

See Also Compiled PLCs (Writing a PLC Program)
I-variable I5
On-line commands **DISABLE PLCC**, **ENABLE PLCC**, **CLEAR**

DELETE ROTARY

Function Delete rotary motion program buffer of addressed coordinate system

Scope Coordinate-system specific

Syntax
```
DELETE ROTARY
DEL ROT
```

Remarks This command causes PMAC to erase the rotary buffer for the currently addressed coordinate system and frees the memory that had been allocated for it.

> ***Note:***
>
> PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if the ROTARY buffer for this coordinate system is open or executing, or if any ROTARY buffer exists for a lower numbered coordinate system, or if a GATHER buffer exists. Any of these buffers must be deleted first. ROTARY buffers must be defined from high-numbered coordinate system to low-numbered coordinate system, and deleted from low-numbered motor to high-numbered motor.

Example
```
&2 DELETE ROTARY          ; Try to erase C.S. 2's rotary buffer
ERR003.....               ; PMAC rejects this; C.S. 1 still has a rotary
&1 DELETE ROTARY          ; Erase C.S. 1's rotary buffer
&2 DELETE ROTARY          ; Erase C.S. 2's rotary buffer; OK now
```

See Also Rotary Program Buffers (Writing a Motion Program)
On-line commands **DEFINE ROTARY**, **OPEN ROTARY**.

DELETE TBUF

Function Delete buffer for axis transformation matrices.

Scope Global

Syntax
```
DELETE TBUF
DEL TBUF
```

Remarks This command frees up the space in PMAC's memory that was used for axis transformation matrices. These matrices can be used for real-time translation, rotation, scaling, and mirroring of the X, Y, and Z axes of any coordinate system.

PMAC will reject this command, reporting an ERR007 if I6=1 or 3, if any ROTARYor GATHER buffer exists. Any of these buffers must be DELETEd first.

Example
```
DEL TBUF
DELETE TBUF
```

See Also Axis Transformation Matrices (Writing a Motion Program)
On-line commands **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **AROT**, **IDIS**, **IROT**, **TINIT**

DELETE TCOMP

Function Erase torque compensation table

Scope Motor specific

Syntax
```
DELETE TCOMP
DEL TCOMP
```

Remarks This command causes PMAC to erase the torque compensation table for the addressed motor, freeing that memory for other use.

PMAC will reject this command, reporting an ERR003 if I6=1 or 3, if any TCOMP buffer exists for a lower numbered motor, or if any BLCOMP, TBUF, ROTARY, or GATHER buffer exists. Any of these buffers must be deleted first. TCOMP buffers must be defined from high-numbered motor to low-numbered motor, and deleted from low-numbered motor to high-numbered motor.

Example
```
#2 DEL TCOMP              ; Erase table belonging to Motor 2
ERR003.....               ; PMAC rejects this command
#1 DEL TCOMP              ; Erase table belonging to Motor 1
#2 DEL TCOMP              ; Erase table belonging to Motor 2
```

See Also Torque Compensation (Setting Up a Motor)
I-variables I51
On-line command **DEFINE TCOMP**

DELETE TRACE

Function Formerly: Erase the motion program trace buffer.

Scope Global

Syntax
```
DELETE TRACE
DEL TRAC
```

Remarks The TRACE buffer feature on PMAC has been removed. **DELETE TRACE** is still a valid command and will not cause an error when sent to PMAC, but it causes no operation to be performed.

Example
```
CLOSE.....                ; Make sure no buffers are open
DELETE GATHER DELETE TRACE ; Free memory
OPEN PLC 17.              ; Open new buffer for entry
CLEAR.....                ; Erase contents of buffer
...                       ; Enter new contents here
```

See Also On-line commands **DELETE GATHER**.

DISABLE PLC

Function Pause execution of specified PLC program(s).

Scope Global

Syntax
```
DISABLE PLC {constant}[,{constant}...]
DIS PLC {constant}[,{constant}...]
DISABLE PLC {constant}[..{constant}]
DIS PLC {constant}[..{constant}]
```
where:

- **{constant}** is an integer from 0 to 31, representing the program number

Remarks This command causes PMAC to disable (stop executing) the specified uncompiled PLC program or programs. Execution can subsequently be resumed at the top of the program with the **ENABLE PLC** command. If it is desired to restart execution at the stopped point, execution should be stopped with the **PAUSE PLC** command, and restarted with the **RESUME PLC** command

The on-line **DISABLE PLC** command can only suspend execution of a PLC program at the end of a scan, which is either the end of the program, or after an **ENDWHILE** statement in the program.

PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs. PLC programs can be re-enabled by using the **ENABLE PLC** command.

If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.

Example
```
DISABLE PLC 1
DIS PLC 5
DIS PLC 3,4,7
DISABLE PLC 0..31
```

See Also I-variable I5
On-line commands **ENABLE PLC**, **OPEN PLC**, **PAUSE PLC**, **RESUME PLC**, **LIST PLC**, **<CONTROL-D>**.
Program commands **DISABLE PLC**, **ENABLE PLC**, **PAUSE PLC**, **RESUME PLC**

DISABLE PLCC

Function Disable compiled PLC program(s).

Scope Global

Syntax
```
DISABLE PLCC {constant}[,{constant}]
DIS PLCC {constant}[,{constant}]
DISABLE PLCC {constant}..{constant}
DIS PLCC {constant}..{constant}
```
where:

- **{constant}** is an integer from 0 to 31, representing the compiled PLC program number

Remarks This command causes PMAC to disable (stop executing) the specified compiled PLC program or programs. Compiled PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs. PLC programs can be re-enabled by using the **ENABLE PLCC** command.

If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.

Example
```
DISABLE PLCC 1
DIS PLCC 5
DIS PLCC 3,4,7
DISABLE PLCC 0..31
```

See Also I-variable I5
On-line commands **DISABLE PLC**, **ENABLE PLC**, **ENABLE PLCC**, **OPEN PLC**, **<CONTROL-D>**.
Program commands **DISABLE PLC**, **DISABLE PLCC**, **ENABLE PLC**, **ENABLE PLCC**

EAVERSION

Function Report full specification of firmware version

Scope Global

Syntax **EAVERSION**
EAVER

Remarks This command causes PMAC to report the full version of the firmware version that it is using. It always returns an 8-digit response, with the following meanings to the digits.

Digit #	Meaning	Settings
1	PMAC Type	0 = PMAC(1) 1 = PMAC2 2 = Ultralite PMAC 3 = PMAC2-VME 4 = Ultralite PMAC2-VME
2	Memory backup type	0 = Battery backup 1 = AMD-style flash backup 2 = Intel-style flash backup
3	Options	0 = Standard 1 = ESA 2 = Lookahead 3 = ESA + Lookahead
4	Test version	0 = Released 1 = 1st test version 2 = 2nd test version, etc.
5	Revision suffix	0 = First released version of revision number 1 = A version 2 = B version, etc.
6 – 8	Released version number	3-digit representation without decimal point (e.g. 116 for 1.16)

Example
```
EAVERSION
01007116                ; PMAC(1), AMD-flash backup,
                        ; Standard firmware, released,
                        ; G revision of 1.16
EAVER
41138116                ; Ultralite PMAC2-VME, AMD-flash,
                        ; ESA firmware, 3rd test version,
                        ; H revision of 1.16
```

See Also On-line commands **DATE**, **TYPE**, **VERSION**

ENABLE PLC

Function Enable specified PLC program(s).

Scope Global

Syntax **ENABLE PLC {constant}[,{constant}...]**
ENA PLC {constant}[,{constant}...]
ENABLE PLC {constant}[..{constant}]
ENA PLC {constant}[..{constant}]
where:

- **{constant}** is an integer from 0 to 31, representing the program number

Remarks This command causes PMAC to enable (start executing) the specified uncompiled PLC program or programs at the top of the program. Execution of the PLC program may have been stopped with the **DISABLE PLC**, **PAUSE PLC**, or **OPEN PLC** command.

PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.

If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.

I-variable I5 must be in the proper state to allow the PLC program(s) specified in this command to execute.

> ***Note:***
>
> This command must be used to start operation of a PLC program after it has been entered or edited, because the **OPEN PLC** command automatically disables the program, and **CLOSE** does not re-enable it.

Example

```
ENABLE PLC 1
ENA PLC 2,7
ENABLE PLC 3,21
ENABLE PLC 0..31
```

This example shows the sequence of commands to download a very simple PLC program and have it enabled automatically on the download:

```
OPEN PLC 7 CLEAR
P1=P1+1
CLOSE
ENABLE PLC 7
```

See Also I-variable I5
On-line commands **DISABLE PLC**, **OPEN PLC**, **PAUSE PLC**, **RESUME PLC**, **LIST PLC**, **<CONTROL-D>**.
Program commands **DISABLE PLC**, **ENABLE PLC**, **PAUSE PLC**, **RESUME PLC**

ENABLE PLCC

Function Enable specified compiled PLC program(s).

Scope Global

Syntax

```
ENABLE PLCC {constant}[,{constant}]
ENA PLCC {constant}[,{constant}]
ENABLE PLCC {constant}..{constant}
ENA PLCC {constant}..{constant}
```

where:

- **{constant}** is an integer from 0 to 31, representing the program number

Remarks This command causes PMAC to enable (start executing) the specified compiled PLC program or programs. Compiled PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.

If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.

I-variable I5 must be in the proper state to allow the compiled PLC program(s) specified in this command to execute.

Example
```
ENABLE PLCC 1
ENA PLCC 2,7
ENABLE PLCC 3,21
ENABLE PLCC 0..31
```

See Also I-variable I5
On-line commands **DISABLE PLC**, **DISABLE PLCC**, **ENABLE PLC**, **OPEN PLC**, **<CONTROL-D>**.
Program commands **DISABLE PLC**, **DISABLE PLCC**, **ENABLE PLC**, **ENABLE PLCC**

ENDGATHER

Function Stop data gathering.

Scope Global

Syntax
```
ENDGATHER
ENDG
```

Remarks This command causes data gathering to cease. Data gathering may start up again (without overwriting old data) with another **GATHER** command.
This command is usually used in conjunction with the data gathering and plotting functions of the PMAC Executive Program.
If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.

Example
```
GAT B1R ..... ; Start gathering and run program 1
ENDG..... Stop gathering – give this command when moves
..... of interest are done
OPEN PROG2 CLEAR
X10
DWELL1000
CMD"GATHER". ; Program issues start command here
X20..... ; Move of interest
DWELL50
CMD"ENDG"..... ; Program issues stop command here
CLOSE
```

See Also Data Gathering Function (Analysis Features)
I-variables I19-I44
On-line commands **DEFINE GATHER**, **GATHER**, **LIST GATHER**, **DELETE GATHER**
Gathering and Plotting (PMAC Executive Program Manual)

F

Function Report motor following error

Scope Motor specific

Syntax **F**

Remarks This command causes PMAC to report the present motor following error (in counts, rounded to the nearest tenth of a count) to the host. Following error is the difference between motor desired and measured position at any instant. When the motor is open-loop (killed or enabled), following error does not exist and PMAC reports a value of 0.

Example
```
F..... ; Ask for following error of addressed motor
12 ..... ; PMAC responds
#3F..... ; Ask for following error of Motor 3
-6.7 ; PMAC responds
```

See Also Following Error (Servo Features)
I-variables Ix11, Ix12, Ix67
On-line commands **<CTRL-F>**, **P**, **V**
Suggested M-variable definitions Mx61, Mx62
Memory map registers D:$0028, D:$002C, etc.; D:$0840, etc.

FRAX

Function Specify the coordinate system's feedrate axes.

Scope Coordinate-system specific

Syntax **FRAX**
FRAX({axis}[,{axis}...])
where:

- **{axis}** (optional) is a character (X, Y, Z, A, B, C, U, V, W) specifying which axis is to be used in the vector feedrate calculations

> ***Note:***
> No spaces are permitted in this command.

Remarks This command specifies which axes are to be involved in the vector-feedrate (velocity) calculations for upcoming feedrate-specified (**F**) moves. PMAC calculates the time for these moves as the vector distance (square root of the sum of the squares of the axis distances) of all the feedrate axes divided by the feedrate. Any non-feedrate axes commanded on the same line will complete in the same amount of time, moving at whatever speed is necessary to cover the distance in that time.

Vector feedrate has obvious geometrical meaning only in a Cartesian system, for which it results in constant tool speed regardless of direction, but it is possible to specify for non-Cartesian systems, and for more than three axes.

> ***Note:***
> If only non-feedrate axes are commanded to move in a feedrate-specified move, PMAC will consider the axis or axes commanded to be feedrate axes for that move. However, the units of these axes may be completely different from those of the vector feedrate axes (e.g. degrees instead of mm), so the speed may not be what is desired.

If a motion program buffer is open when this command is sent to PMAC, it will be entered into the buffer for later execution.

For instance, in a Cartesian XYZ system, if you use **FRAX(X,Y)**, all of your feedrate-specified moves will be at the specified vector feedrate in the XY-plane, but not necessarily in XYZ-space. If you use **FRAX(X,Y,Z)** or **FRAX**, your feedrate-specified moves will be at the specified vector feedrate in XYZ-space. Default feedrate axes for a coordinate system are X, Y, and Z.

Example

```
FRAX.....          ; Make all axes feedrate axes
FRAX(X,Y) .....    ; Make X and Y axes only the feedrate axes
FRAX(X,Y,Z)        ; Make X, Y, and Z axes only the feedrate axes
```

See Also Feedrate-Specified Moves (Writing a Motion Program)
Program commands **F{data}**, **FRAX**.

GATHER

Function Begin data gathering.

Scope Global

Syntax
```
GATHER [TRIGGER]
GAT [TRIG]
```

Remarks This command causes data gathering to commence according to the configuration defined in I-variables I19-I45. If **TRIGGER** is not used in the command, gathering will start on the next servo cycle. If **TRIGGER** is used, gathering will start on the first servo cycle after machine input MI2 goes true.

Gathering will proceed at the frequency set by I19 (in number of servo interrupt cycles). If I19 is 0, only one set of data will be gathered per **GATHER** command. If PMAC is already gathering data, **GATHER** will cause resynchronization of the gathering cycle to the next servo cycle.

Gathering will continue until PMAC receives an **ENDGATHER** command, or until the buffer created by the **DEFINE GATHER** command is full.

This command is usually used in conjunction with the data gathering and plotting functions of the PMAC Executive Program.

If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.

Example
```
GAT B1R .....          Start gathering and run program 1
ENDG.....              Stop gathering – give this command when moves
.....                   of interest are done
OPEN PROG2 CLEAR
X10
DWELL1000
CMD"GATHER".           Program issues start command here
X20 .....              Move of interest
DWELL50
CMD"ENDG".....         Program issues stop command here
CLOSE
```

See Also Data Gathering Function (Analysis Features)
I-variables I19-I45
On-line commands **DEFINE GATHER**, **GATHER**, **LIST GATHER**, **DELETE GATHER**
Gathering and Plotting (PMAC Executive Program Manual)

H

Function Perform a feedhold

Scope Coordinate-system specific

Syntax **H**

Remarks This causes the currently addressed coordinate system to suspend execution of the program starting immediately by bringing its time base value to zero, decelerating along its path at a rate defined by the coordinate system I-variable Ix95. Technically the program is still executing after an **H** command, but at zero speed. This means that the motors defined in the coordinate system cannot be moved while performing the feed hold.

To do a hold of the currently addressed coordinate system in a manner that permits jogging of the motors in the coordinate system while in feed hold mode, refer to the **** "program hold" command.

The **H** command is very similar in effect to a **%0** command, except that deceleration is controlled by Ix95, not Ix94, and execution can be resumed with an **R** or an **S** command, instead of a **%100** command. In addition, **H** works under external time base, whereas a **%0** command does not.

Full speed execution along the path will commence again on an **R** or **S** command. The ramp up to full speed will also take place at a rate determined by Ix95 (full time-base value, either internally or externally set). Once the full speed is reached, Ix94 determines any time-base changes.

See Also Stopping Commands (Making Your Application Safe)
Control-Panel Port HOLD/ Input (Connecting PMAC to the Machine)
Time-Base Control (Synchronizing PMAC to External Events)
I-variables I52, Ix93, Ix94, Ix95
On-line commands **<CTRL-O>**, **%**, **%{constant}**, **A**, **K**, ****, **Q**
JPAN Connector Pin 12

HOME

Function Start Homing Search Move

Scope Motor specific

Syntax **HOME**
HM

Remarks This command causes the addressed motor to perform a homing search routine. The characteristics of the homing search move are controlled by motor I-variables Ix03 and Ix19-Ix26, plus encoder I-variables 2 and 3 for that motor's position encoder.

The on-line home command simply starts the homing search routine. PMAC provides no automatic indication that the search has completed (although the In-Position interrupt could be used for this purpose) or whether the move completed successfully. Polling, or a combination of polling and interrupts, is generally used to determine completion and success.

By contrast, when a homing search move is given in a motion program (e.g. **HOME1,2**), the motion program will keep track of completion by itself as part of its sequencing algorithms.

If there is an axis offset in the axis-definition statement for the motor, and/or following error in the motor servo loop, the reported position at the end of the homing search move will be equal to the axis offset minus the following error, not to zero.

Example

```
HOME.....          ; Start homing search on the addressed motor
#1HM.....          ; Start homing search on Motor 1
#3HM#4HM           ; Start homing search on Motors 3 and 4
```

See Also Control Panel Port HOME/ Input (Connecting PMAC to the Machine)
Homing Moves (Basic Motor Moves)
I-variables Ix03, Ix19-Ix26, Encoder I-Variables 2 & 3
On-line command **HOMEZ**
Program command **HOME{constant}**, **HOMEZ{constant}**
JPAN Connector Pin 11

HOMEZ

Function Do a Zero-Move Homing

Scope Motor specific

Syntax **HOMEZ**
HMZ

Remarks This command causes the addressed motor to perform a zero-move homing search. Instead of jogging until it finds a pre-defined trigger, and calling its position at the trigger the home position, with this command, the motor calls wherever it is (*commanded* position) at the time of the command the home position.

If there is an axis offset in the axis-definition statement for the motor, and/or following error in the motor servo loop, the reported position at the end of the homing operation will be equal to the axis offset minus the following error, not to zero.

Example

```
; On-line command examples
HOMEZ ..............    ; Do zero-move homing search on the addressed motor
#1HMZ ..............    ; Do zero-move homing search on Motor 1
#3HMZ#4HMZ ...          ; Do zero-move  homing search on Motors 3 and 4
 ..........................  ; Buffered motion program examples
HOMEZ1
HOMEZ2,3
 ..........................  ; On-line commands issued from PLC program
IF (P1=1) ......        ;
  CMD"#5HOMEZ"          ; Program issues on-line command
  P1=0 .............    ; So command is not repeatedly issued
ENDIF
```

See Also Homing Moves (Basic Motor Moves)
On-line command **HOME**
Program command **HOME{constant}**, **HOMEZ{constant}**

I{constant}

Function Report the current I-variable value(s).

Scope Global

Syntax **I{constant}[..{constant}]**

where:

- **{constant}** is an integer from 0 to 1023 representing the number of the I-variable; the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;

Remarks This command causes PMAC to report the current value of the specified I-variable or range of I-variables.

When is 0 or 2, only the value of the I-variable itself is returned (e.g. *10000*). When I9 is 1 or 3, the entire variable value assignment statement (e.g. *I130=10000*) is returned by PMAC.

When I9 is 0 or 1, the values of "address" I-variables are reported in decimal form. When I9 is 2 or 3, the values of these variables are reported in hexadecimal form.

> ***Note:***
> If a motion program buffer (including a rotary buffer) is open, **I{constant}** will be entered into that buffer for later execution, to be interpreted as a full-circle move command with a vector to the center along the X-axis (see Circular Moves in the Writing a Motion Program section).

Example

```
I5 ..................   ; Request the value of I5
2 ....................   ; PMAC responds
I130..135 ............   ; Request the value of I130 through I135
60000 ................   ; PMAC responds with 6 lines
5000
5000
50000
1
20000
```

To see the effect of I9 on the form of the response, observe the following:

```
I9=0   I125
49152 ................   ; Short form, decimal
I9=1   I125
I125=49152 ...           ; Long form, decimal
I9=2   I125
$C000 ................   ; Short form, hexadecimal
I9=3   I125
I125=$C000               ; Long form, hexadecimal
```

See Also

Initialization (I) Variables (Computational Features)
I-Variable Specifications
I-variable I9
On-line commands **I{constant}={expression}**, **M{constant}**, **P{constant}**, **Q{constant}**
Program commands **{axis}{data}{vector}{data}**, **I{data}**

I{constant}={expression}

Function Assign a value to an I-variable.

Scope Global

Syntax **I{constant}[..{constant}]={expression}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the I-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{expression}** contains the value to be given to the specified I-variable(s)

Remarks This command assigns the value on the right side of the equals sign to the specified I-variable or range of I-variables.

If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into the buffer for later execution.

Example

```
I5=2
I130=1.25*I130
I22..44=0
I102=$C003
I104=I103
```

See Also Initialization (I) Variables (Computational Features)
I-Variable Specifications
On-line commands **I{constant}**, **M{constant}={expression}**, **P{constant}={expression}**, **Q{constant}={expression}**

I{constant}=*

Function Assign factory default value to an I-variable.

Scope Global

Syntax **I{constant}[..{constant}]=***
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the I-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range.

Remarks This command sets the specified I-variable or range of I-variables to the factor default value. Each I-variable has its own factory default; these are shown in the I-Variable Specification section.

Example
```
I13=*
I100..199=*
```

See Also Initialization (I) Variables (Computational Features)
I-Variable Specifications – Default Values
On-line commands **I{constant}**, **I{constant}={expression}**

INC

Function Specify Incremental Move Mode

Scope Coordinate-system specific

Syntax
```
INC
INC({axis}[,{axis}...])
```
where:

- **{axis}** is a letter (X, Y, Z, A, B, C, U, V, W) representing the axis to be specified, or the character R to specify radial vector mode

> ***Note:***
> No spaces are permitted in this command.

Remarks The **INC** command without arguments causes all subsequent positions for all axes in position motion commands to be treated as incremental distances. An **INC** statement with arguments causes the specified axes to be in incremental mode, and all others stay the way they were before. The default axis specification is absolute.

If 'R' is specified as one of the 'axes', the I, J, and K terms of the circular move radius vector specification will be specified in incremental form (i.e. as a vector from the move start point, not from the origin). An **INC** command without any arguments does not affect this vector specification. The default vector specification is incremental.

If a motion program buffer is open when this command is sent to PMAC, it will be entered into the buffer as a program statement.

Example
```
INC(A,B,C) .          ; A, B, & C axes made incremental – other axes and rad vector left as is
INC.....              ; All axes made incremental – radius vector left as is
INC(R)                ; Radius vector made incremental – all axes left as is
```

See Also Circular Moves (Writing a Motion Program)
On-line command **ABS**
Program commands **ABS**, **INC**

J!

Function Adjust motor commanded position to nearest integer count

Scope Motor specific

Syntax **J!**

Remarks This command causes the addressed motor, if the desired velocity is zero, to adjust its commanded position to the nearest integer count value. It can be valuable to stop dithering if the motor is stopped with its commanded position at a fractional value, and integral gain is causing oscillation about the commanded position.

Example

```
OPEN PLC 7 CLEAR
IF (M50=1) .                ; Condition to start branch
     CMD"#1J/"              ; Tell motor to stop
     WHILE (M133=0)         ; Wait for desired velocity to reach zero
     ENDWHILE
     CMD"#1J!"              ; Adjust command position to integer value
     M50=0 .                ; To keep from repeated execution
ENDIF
```

See Also On-line commands **J/**, **J={constant}**

J+

Function Jog Positive

Scope Motor specific

Syntax **J+**

Remarks This command causes the addressed motor to jog in the positive direction indefinitely. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
J+ .....                    ; Jog addressed motor positive
#7J+.....                   ; Jog Motor 7 positive
#2J+#3J+                    ; Jog Motors 2 and 3 positive
```

See Also Control Panel Port JOG+/ Input (Connecting PMAC to the Machine)
JPAN Connector Pin 6
Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
On-line commands **J-**, **J/**, **J=**, **J={constant}**, **J:{constant}**, **J^{constant}**

J-

Function Jog Negative

Scope Motor specific

Syntax **J-**

Remarks This command causes the addressed motor to jog in the negative direction indefinitely. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
J- ..... ; Jog addressed motor negative
#5J-..... ; Jog Motor 5 negative
#3J-#4J- ; Jog Motors 3 and 4 negative
```

See Also Control Panel Port JOG-/ Input (Connecting PMAC to the Machine)
JPAN Connector Pin 4
Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
On-line commands **J+**, **J/**, **J=**, **J={constant}**, **J:{constant}**, **J^{constant}**

J/

Function Jog Stop

Scope Motor specific

Syntax **J/**

Remarks This command causes the addressed motor to stop jogging. It also restores position control if the motor's servo loop has been opened (enabled or killed), with the new commanded position set equal to the actual position at the time of the **J/** command. Jogging deceleration is determined by the values of Ix19-Ix21 in force at the time of this command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
#1J+..... ; Jog Motor 1 positive
J/ ..... ; Stop jogging Motor 1
O5 ..... ; Open-loop output of 5% on Motor 1
O0 ..... ; Open loop output of 0%
J/ ..... ; Restore closed-loop control
K..... ; Kill output
J/ ; Restore closed-loop control
```

See Also Control Panel Port JOG+/, JOG-/ Inputs (Connecting PMAC to the Machine)
JPAN Connector Pin 4, 6
Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
On-line commands **<CTRL-A>**, **A**, **J+**, **J-**, **J=**, **J={constant}**, **J:{constant}**, **J^{constant}**, **K**, **O{constant}**

J:{constant}

Function Jog Relative to Commanded Position

Scope Motor specific

Syntax **J:{constant}**
where:

- **{constant}** is a floating point value specifying the distance to jog, in counts.

Remarks This command causes a motor to jog the distance specified by **{constant}** relative to the present commanded position. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command. Compare to **J^{constant}**, which is a jog relative to the present actual position. A variable incremental jog command can be executed with the **J:*** command. PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
#1HM.....                ; Do homing search move on Motor 1
J:2000.....              ; Jog a distance of 2000 counts (to 2000 counts)
J:2000                   ; Jog a distance of 2000 counts (to 4000 counts)
```

See Also Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
On-line commands **J+**, **J-**, **J/**, **J=**, **J={constant}**, **J^{constant}**

J:*

Function Jog to specified variable distance from present commanded position

Scope Motor specific

Syntax **J:***

Remarks This command causes the addressed motor to jog the distance specified in the motor's variable jog position/distance register relative to the present commanded position. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command. Compare to **J^*** , which is a jog relative to the present actual position.

The variable jog position/distance register is a floating-point register with units of counts. It is best accessed with a floating-point M-variable. The register is located at PMAC address L:$082B for motor 1, L:$08EB for motor 2, etc. (suggested M-variable Mx72) The usual procedure is to write the destination position to this register by assigning a value to the M-variable, then issuing the **J:*** command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
M172->L:$082B            ; Define #1 variable jog position/distance reg.
#1HMZ .....              ; Declare present position to be zero
M172=3000.....           ; Assign distance value to register
#1J:* .....              ; Jog Motor 1 this distance; end cmd. pos. will be 3000
#1J:* .....              ; Jog Motor 1 this distance; end cmd. pos. will be 6000
M172=P1*SIN(P2)          ; Assign new distance value to register
#1J:* .....              ; Jog Motor 1 this distance
#1J=                     ; Return to prejog target position
```

See Also Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
Memory map registers L:$082B, L:$08EB, etc.
Suggested M-variable definitions M172, M272, etc.
On-line commands **J=**, **J={constant}**, **J=***, **J^***

J=

Function Jog to Prejog Position

Scope Motor specific

Syntax **J=**

Remarks This command causes the addressed motor to jog to the last pre-jog and pre-handwheel-move position (the most recent programmed position). Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.

The register containing this position information for the motor is called the target position register (D:$080B for Motor 1, D:$08CB for Motor 2, etc.). Suggested M-variable definitions M163, M263, etc. can be used in programs to give access to these registers.

If the **/** or **** stop command has been used to suspend program execution, and one or more motors jogged away from the stop position, the **J=** command must be used to return the motor(s) back to the stop position before program execution can be resumed.

The **J=** command can also be useful if a program has been aborted in the middle of a move, because it will move the motor to the programmed move end position (provided I13=0 so PMAC is not in segmentation mode), so the program may be resumed properly from that point.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
&1Q.....          ; Stop motion program at end of move
#1J+.....         ; Jog Motor 1 away from this position
J/ .....          ; Stop jogging
J= .....          ; Jog back to position where program quit
R.....            ; Resume motion program

&1A.....          ; Stop motion program in middle of move
#1J=#2J=#3J=      ; Move all motors to original move end position
R                 ; Resume motion program
```

See Also Control Panel Port PREJ/ Input (Connecting PMAC to the Machine)
JPAN Connector Pin 7
Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
On-line commands **J+**, **J-**, **J/**, **J={constant}**, **J:{constant}**, **J^{constant}**

J={constant}

Function Jog to specified position

Scope Motor specific

Syntax **J={constant}**
where:

- **{constant}** is a floating point value specifying the location to which to jog, in encoder counts.

Remarks This command causes the addressed motor to jog to the position specified by **{constant}**. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.

A variable jog-to-position can be executed with the **J=*** command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
J=0 ..... ; Jog addressed motor to position 0
#4J=5000 ..... ; Jog Motor 4 to 5000 counts
#8J=-32000 ; Jog Motor 8 to -32000 counts
```

See Also Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
On-line commands **J+**, **J-**, **J/**, **J=**, **J:{constant}**, **J^{constant}**, **J=***, **J:***, **J^***

J=*

Function Jog to specified variable position

Scope Motor specific

Syntax **J=***

Remarks This command causes the addressed motor to jog to the position specified in the motor's variable jog position/distance register. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.

The variable jog position/distance register is a floating-point register with units of counts. It is best accessed with a floating-point M-variable. The register is located at PMAC address L:$082B for motor 1, L:$08EB for motor 2, etc. (suggested M-variable Mx72). The usual procedure is to write the destination position to this register by assigning a value to the M-variable, then issuing the **J=*** command.

Virtually the same result can be obtained by writing to the motor target position register and issuing the **J=** command. However, using the **J=*** command permits you to return to the real target position afterwards without having to restore the target position register. Also, the **J=*** command uses a register whose value is scaled in counts, not fractions of a count.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
M172->L:$082B ; Define #1 variable jog position/distance reg.

M172=3000..... ; Assign position value to register
#1J=* ..... ; Jog Motor 1 to this position
M172=P1*SIN(P2) ; Assign new position value to register
#1J=* ..... ; Jog Motor 1 to this position
#1J= ; Return to prejog target position
```

See Also Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
Memory map registers L:$082B, L:$08EB, etc.
Suggested M-variable definitions M172, M272, etc.
On-line commands **J=**, **J={constant}**, **J:***, **J^***

J=={constant}

Function Jog to specified motor position and make that position the "pre-jog" position

Scope Motor specific

Syntax **J=={constant}**
where:

- **{constant}** is a floating point value specifying the location to which to jog, in encoder counts.

Remarks This command causes the addressed motor to jog the position specified by **{constant}**. It also makes this position the pre-jog position, so it will be the destination of subsequent **J=** commands. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
#1J==10000 .          ; Jog Motor 1 to 10000 counts and make that the pre-jog
.....                 ; position.
J+ .....              ; Jog indefinitely in the positive direction
J=                    ; Return to 10000 counts
```

See Also Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
On-line commands **J=**, **J={constant}**, **J=***, **J^***

J^{constant}

Function Jog Relative to Actual Position

Scope Motor specific

Syntax **J^{constant}**
where:

- **{constant}** is a floating point value specifying the distance to jog, in counts.

Remarks This causes a motor to jog the distance specified by **{constant}** relative to the present actual position. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command. Compare to **J:{constant}**, which is a jog relative to the present commanded position.

Usually the **J:{constant}** command is more useful, because its destination is not dependent on the following error at the instant of the command. The **J^0** command can be useful for "swallowing" any existing following error.

A variable incremental jog can be executed with the **J^*** command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
#1HM.....             ; Do homing search move on Motor 1
J^2000.....           ; Jog a distance of 2000 counts from actual position
.....                 ; If actual was -5 cts, new command pos is 1995 cts
J^2000.....           ; Jog a distance of 2000 counts from actual position
                      ; If actual was 1992 cts, new cmd pos is 3992 cts
```

See Also Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22

On-line commands **J+**, **J-**, **J/**, **J=**, **J={constant}**, **J:{constant}**, **J=***, **J:***, **J^***

J^*

Function Jog to specified variable distance from present actual position

Scope Motor specific

Syntax **J^***

Remarks This command causes the addressed motor to jog the distance specified in the motor's variable jog position/distance register relative to the present actual position. Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command. Compare to **J:*** , which is a jog relative to the present commanded position.

The variable jog position/distance register is a floating-point register with units of counts. It is best accessed with a floating-point M-variable. The register is located at PMAC address L:$082B for motor 1, L:$08EB for motor 2, etc. (suggested M-variable Mx72).The usual procedure is to write the destination position to this register by assigning a value to the M-variable, then issuing the **J^*** command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
M172->L:$082B          ; Define #1 variable jog position/distance reg.
#1HMZ ..............   ; Declare present position to be zero
M172=3000.....         ; Assign distance value to register
#1J^* ..............   ; Jog Motor 1 this distance;  if following error at
 .....                 ; command was 3, end cmd. pos. will be 2997
#1J^* ..............   ; Jog Motor 1 this distance; if following error at
 .....                 ; command was 2, end cmd. pos. will be 5995
M172=P1*SIN(P2)        ; Assign new distance value to register
#1J^* ..............   ; Jog Motor 1 this distance
#1J=                   ; Return to prejog target position
```

See Also Jogging Moves (Basic Motor Moves)
I-variables Ix19-Ix22
Memory map registers L:$082B, L:$08EB, etc.
Suggested M-variable definitions M172, M272, etc.
On-line commands **J=**, **J={constant}**, **J=***, **J^***

{jog command}^{constant}

Function Jog until trigger

Scope Motor specific

Syntax

```
J=^{constant}
J={constant}^{constant}
J:{constant}^{constant}
J^{constant}^{constant}
J=*^{constant}
J:*^{constant}
J^*^{constant}
```

where:

- **{constant}** after the ^ is a floating point value specifying the distance from the trigger to which to jog after the trigger is found, in encoder counts

Remarks This command format permits a jog-until-trigger function. When the **^{constant}** structure is added to any definite jog command, the jog move can be interrupted by a pre-defined trigger condition, and the motor will move to a point relative to the trigger position as specified by the final value in the command.

The indefinite jog commands **J+** and **J-** cannot be turned into jog-until-trigger moves.

Jog-until-trigger moves are very similar to homing search moves, except they have a definite end position in the absence of a trigger, and they do not change the motor zero position. In the absence of a trigger, the move will simply stop at the pre-defined position.

The trigger condition for a jog-until-trigger move can be either an input flag, or a warning following error condition for the motor. If bit 17 of Ix03 is 0 (the default), the trigger is a transition of an input flag and/or encoder index channel from the set defined for the motor by Ix25. Encoder/flag variables 2 and 3 (e.g. I912 and I913) define which edges of which input signals create the trigger.

If bit 17 of Ix03 is 1, the trigger is the warning following error status bit of the motor becoming true. Ix12 for the motor sets the error threshold for this condition.

The trigger position can either be the hardware-captured position, or a software-read position. If bit 16 of Ix03 is 0 (the default), the encoder position latched by the trigger in PMAC's DSPGATE hardware is used as the trigger position. This is the most accurate option because it uses the position at the moment of the trigger, but it can only be used with incremental encoder feedback brought in on the same channel number as the triggering flag set. This option cannot be used for other types of feedback, or for triggering on following error.

If bit 16 of Ix03 is 1, PMAC reads the present sensor position after it sees the trigger. This can be used with any type of feedback and either trigger condition, but can be less accurate than the hardware capture because of software delays.

Jogging acceleration and velocity are determined by the values of Ix19-Ix22 in force at the time of this command.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
#1J=^1000 ..          ; Jog to pre-jog position in the absence of a trigger, but if trigger
.....                 ; is found, jog to +1000 counts from trigger.
#2J:5000^-100         ; Jog 5000 counts in the positive direction in the absence of a
.....                 ; trigger, but if trigger is found, jog to -100 counts from
.....                 ; trigger position.
#3J=20000^0.          ; Jog to 20000 counts in the absence of a trigger, but if
                      ; trigger is found, return to trigger position.
```

See Also Jogging Moves (Basic Motor Moves)
I-variables Ix03, Ix19-Ix22, Ix25, Encoder/Flag I-variables 2 and 3
On-line commands **J=**, **J={constant}**, **J:{constant}**, **J^{constant}**,
.....**J=***, **J:***, **J^***
Program commands **{axis}{data}^{data}**

K

Function Kill motor output

Scope Motor specific

Syntax **K**

Remarks This command causes PMAC to kill the outputs for the addressed motor. The servo loop is disabled, the DAC outputs are set to zero (Ix29 and/or Ix79 offsets are still in effect), and the AENA output for the motor is taken to the disable state (polarity is determined by E17).

Closed-loop control of this motor can be resumed with a **J** command. The **A** command will re-establish closed-loop control for all motors in the addressed coordinate system, and the **<CTRL-A>** command will do so for all motors on PMAC.

The action on a **K** command is equivalent to what PMAC does automatically to the motor on an amplifier fault or a fatal following error fault.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3). The program must be stopped first, usually with an **A** command. However, the global **<CTRL-K>** command will kill all motors immediately, regardless of whether any are running motion programs.

Example

```
K.....          ; Kill the addressed motor
#1K.....        ; Kill Motor 1
J/              ; Re-establish closed-loop control of Motor 1
```

See Also Amplifier Fault, Following Error Limits, Stop Commands (Making Your Application Safe)
I-variables Ix29, Ix79
On-line commands **<CTRL-A>**, **<CTRL-K>**, **A**, **Q**, **H**, **J/**
Jumpers E17, E17A-E17H

LEARN

Function Learn present commanded position

Scope Coordinate-system specific

Syntax

```
LEARN[({axis}[,{axis}...]]
LRN[({axis}[,{axis}...]]
```

> ***Note:***
> No spaces are permitted in this command.

Remarks This command causes PMAC to add a line to the end of the open motion program buffer containing axis position commands equal to the current commanded positions for some or all of the motors defined in the addressed coordinate system. In this way, PMAC can "learn" a sequence of points to be repeated by subsequent execution of the motion program.

PMAC effectively performs a **PMATCH** function, reading motor commanded positions and inverting the axis definition equations to compute axis positions.

If axis names are specified in the **LEARN** command, only position commands for those axes are used in the line added to the motion program. If no axis names are specified in the learn command, position commands for all nine possible axis names are used in the line added to the motion program. The position command for an axis with no motor attached ("phantom" axis) will be zero.

> ***Note:***
>
> If a motor is closed loop, the learned position will differ from the actual position by the amount of the position following error because commanded position is used. If a motor is open-loop or killed, PMAC automatically sets motor commanded position equal to motor actual position, so the **LEARN** function can be used regardless of the state of the motor.

Example

```
&1 .........                          ; Address coordinate system 1
#1->10000X .                         ; Define motor 1 in C.S. 1
#2->10000Y ...                       ; Define motor 2 in C.S. 1
OPEN PROG 1 CLEAR                    ; Prepare program buffer for entry
F10 TA200 TS50                       ; Enter required non-move commands
{Move motors to a position, e.g. #1 to 13450 commanded, #2 to 29317 commanded}
LEARN(X,Y) ...                       ; Tell PMAC to learn these positions
X1.345 Y2.9317                       ; This is the line that PMAC adds to PROG 1
{move motors to new position, e.g. #1 to 16752 cmd., #2 to 34726 cmd}
LEARN .....                          ; Tell PMAC to learn positions
A0 B0 C0 U0 V0 W0 X1.6752 Y3.4726 Z0
                                     ; PMAC adds positions for all axes to PROG 1
```

See Also Learning a Motion Program (Writing a Motion Program)
On-line command **PMATCH**

LIST

Function List the contents of the currently opened buffer

Scope Global

Syntax **LIST**

Remarks This command causes PMAC to report the contents of the currently opened buffer (PLC, PROG, or ROT) to the host. If no buffer is open, PMAC will report an error (ERR003 if I6=1 or 3). Note that what is reported will not include any **OPEN**, **CLEAR**, or **CLOSE** statements (since these are *not* program commands).

You can list an unopened buffer by specifying the buffer name in the list command (e.g. **LIST PROG 1**). See further **LIST** commands, below.

Example

```
OPEN PROG 1                          ; Open buffer for entry
LIST .......                         ; Request listing of open buffer
LINEAR .....                         ; PMAC reports contents of open buffer
F10
X20 Y20
X0 Y0
RETURN
CLOSE ......                         ; Close buffer
LIST .......                         ; Request listing of open buffer
<BELL>ERR003                         ; PMAC reports error because no open buffer
```

See Also On-line commands **OPEN**, **CLOSE**, **LIST PLC**, **LIST PROGRAM**

LIST BLCOMP

Function List contents of addressed motor's backlash compensation table

Scope Motor specific

Syntax **LIST BLCOMP**

Remarks This command causes PMAC to report to the host the contents of the backlash compensation table belonging to the addressed motor. The values are reported in decimal ASCII form, multiple values to a line, with individual values separated by spaces.

The **LIST BLCOMP DEF** command should be used to report the header information for this table.

If there is no table for the addressed motor, PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Example
```
LIST BLCOMP.                              ; Request contents of backlash comp  table
9 17 -3 6 35 87 65 24 18 -9 -16 -34 ; PMAC responds
-7 12 -3 -8 32 44 16 0 -20 -5 0      ; Continued response
```

See Also Backlash Compensation Tables (Setting Up a Motor)
On-line commands **DEFINE BLCOMP**, **DELETE BLCOMP**, **LIST BLCOMP DEF**

LIST BLCOMP DEF

Function List definition of addressed motor's backlash compensation table

Scope Motor specific

Syntax **LIST BLCOMP DEF**

Remarks This command causes PMAC to report to the host the definition of the backlash compensation table that belongs to the addressed motor. The definition reported consists of the two items established in the **DEFINE BLCOMP** command that set up the motor:

1. The number of entries in the table;
2. The span of the table in counts of the motor.

If there is no table for the addressed motor, PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Example
```
LIST BLCOMP DEF                  ; Request def of addressed motor backlash comp table
100,100000 .                     ; PMAC responds; 100 entries in table,
.....                            ; span is 100,000 counts
```

See Also Backlash Compensation Tables (Setting Up a Motor)
On-line commands **DEFINE BLCOMP**, **DELETE BLCOMP**, **LIST BLCOMP**

LIST COMP

Function List contents of addressed motor's compensation table

Scope Motor specific

Syntax **LIST COMP**

Remarks This command causes PMAC to report to the host the contents of the compensation table belonging to the addressed motor. The values are reported in decimal ASCII form, multiple values to a line, with individual values separated by spaces.

The **LIST COMP DEF** command should be used to report the header information for this table.

If there is no table for the addressed motor, PMAC will reject the command (reporting ERR003 if I6=1 or 3).

The compensation table "belonging" to this motor may not affect this motor's position or be affected by it

Example

```
LIST COMP.....                              ; Request contents of compensation table
9 17 -3 6 35 87 65 24 18 -9 -16 -34         ; PMAC responds
-7 12 -3 -8 32 44 16 0 -20 -5 0             ; Continued response
```

See Also Leadscrew Compensation Tables (Setting Up a Motor)
On-line commands **DEFINE COMP**, **DELETE COMP**, **LIST COMP DEF**

LIST COMP DEF

Function List definition of addressed motor's compensation table

Scope Motor specific

Syntax **LIST COMP DEF**

Remarks This command causes PMAC to report to the host the definition of the compensation table that belongs to the addressed motor. The definition reported consists of the four items established in the **DEFINE COMP** command that set up the motor (even if some of those items were not specified explicitly):

1. The number of entries in the table (number of rows and number of columns for a two-dimensional table)
2. The number of the motor whose position provides the source data for the table (both source motors for a two-dimensional table)
3. The number of the motor whose position is modified by the table
4. The span of the table in counts of the source motor (in both dimensions for a two-dimensional table)

Note:

If there is no table for the addressed motor, PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Note:

The compensation table "belonging" to this motor may not affect this motor's position or be affected by it

Example

```
LIST COMP DEF                               ; Request definition of compensation table
100,#2,#2,100000                            ; PMAC responds; 100 entries in table,
.....                                       ; Motor 2 is source and target, span is 100,000
.....                                       ; counts
#3 LIST COMP DEF                            ; Request def of comp table belonging to Motor 3
10,20,#4,#5,#6,50000,100000                 ; PMAC responds
.....                                       ; A 2D 10x20 table, span of 50Kx100K counts
                                            ; #4 & #5 as source, #6 as target
```

See Also Leadscrew Compensation Tables (Setting Up a Motor)
On-line commands **DEFINE COMP**, **DELETE COMP**, **LIST COMP**

LIST GATHER

Function Report contents of the data gathering buffer.

Scope Global

Syntax
```
LIST GATHER [{start}] [,{length}]
LIS GAT [{start}] [,{length}]
```
where:

- The optional **{start}** parameter is an integer constant specifying the distance from the start of the buffer (in words of memory) to begin the listing (0 is the default);
- The optional **{length}** parameter (after a comma) is an integer constant specifying the number of words of the buffer to be sent to the host (to the end of the buffer is the default)

Remarks This command causes PMAC to report the contents of the data-gathering buffer to the host. The data is reported as 48-bit long words in hexadecimal format (12 characters per word) separated by spaces, 16 long words per line.

If neither **{start}** nor **{length}** is specified, the entire contents of the buffer will be reported. If **{start}** is specified, the reporting will begin **{start}** words from the beginning of the buffer. If **{length}** is specified, the reporting will continue for **{length}** words from the starting point.

Example
```
LIST GATHER.                ; reports the whole buffer
LIST GATHER 256             ; skips the first 256 long words
LIST GATHER 0,32            ; reports the first 32 words
LIST GATHER ,32             ; does the same as above
LIST GATHER 64,128          ; skips the first 64 words, reports the next 128
```

See Also Data Gathering Function (Analysis Features)
I-variables I19, I20, I21-I44.
On-line commands **GATHER**, **ENDGATHER**, **DEFINE GATHER**
Gathering and Plotting (PMAC Executive Program Manual)

LIST LINK

Function List Linking Addresses of Internal PMAC Routines

Scope Global

Syntax **LIST LINK**

Remarks This command causes PMAC to list the addresses of the internal routines that the PLC cross-compiler needs to properly compile and link its programs.

This command is used automatically by the PLC cross-compiler in the Executive program.

For the standalone DOS cross-compiler, the ASCII characters of PMAC's response to this command must be contained in a file named LISTLINK.TXT in the same directory and subdirectory as the cross-compiler. Each separate version of PMAC's firmware potentially has different addresses for these routines, so a new LISTLINK.TXT file must be created any time the PMAC firmware is updated, even for a minor change such as from V1.15A to V1.15B.

Example
```
LIST LINK.....                                  ; Request linking addresses
004532 004A97 005619 005F21 0062FE 0063A4 ; PMAC responds
```

See Also Compiled PLCs (Writing a PLC Program)

LIST PC

Function List Program at Program Counter

Scope Coordinate-system specific

Syntax **LIST PC[,[{constant}]]**
where:

- **{constant}** is a positive integer representing the number of words in the program to be listed

Remarks This command causes PMAC to list the program line(s) that it is (are) about to calculate in the addressed coordinate system, with the first line preceded by the program number and each line preceded by the address offset. **LIST PC** just lists the next line to be calculated. **LIST PC,** lists from the next line to be calculated to the end of the program. **LIST PC,{constant}** lists the specified address range size starting at the next line to be calculated. To see the current line of execution, use the **LIST PE** command.

Because PMAC calculates ahead in a continuous sequence of moves, the **LIST PC** (Program Calculation) command will in general return a program line further down in the program than **LIST PE** will.

If the coordinate system is not pointing to any motion program, PMAC will return an error (ERR003 if I6=1 or 3). Initially the pointing must be done with the **B{constant}** command.

Example

```
LIST PC ..........          ; List next line to be calculated
P1:22:X10Y20                ; PMAC responds
LIST PC,4.....              ; List next four words of program to be calculated
P1:22:X10Y20                ; PMAC responds
24:X15Y30
LIST PC, .....              ; List rest of program
P1:22:X10Y20                ; PMAC responds
24:X15Y30
26:M1=0
28:RETURN
```

See Also On-line commands **B{constant}**, **LIST**, **PC**, **LIST PE**, **PE**

LIST PE

Function List Program at Program Execution

Scope Coordinate-system specific

Syntax **LIST PE[,[{constant}]]**
where:

- **{constant}** is a positive integer representing the number of words in the program to be listed

Remarks This command causes PMAC to list the program line(s) starting with the line containing the move that it is currently executing in the addressed coordinate system, with the first line preceded by the program number, and each line preceded by the address offset.

Because PMAC calculates ahead in a continuous sequence of moves, the **LIST PC** (Program Calculation) command will in general return a program line further down in the program than **LIST PE** will.

LIST PE returns only the currently executing line. **LIST PE,** returns from the currently executing line to the end of the program. **LIST PE,{constant}** returns the specified number of words in the program, starting at the currently executing line.

If the coordinate system is not pointing to any motion program, PMAC will return an error (ERR003 if I6=1 or 3). Initially the pointing must be done with the **B{constant}** command.

Example

```
LIST PE ..........      ; List presently executing line
P5:35:X5Y30.            ; PMAC responds
LIST PE,4.....          ; List four program words, starting with executing line
P5:35:X5Y30.            ; PMAC responds
37:X12Y32
LIST PE, ........       ; List rest of program, starting with executing line
P5:35:X5Y30.            ; PMAC responds
37:X12Y32
39:X0 Y10
41:RETURN
```

See Also On-line commands **B{constant}**, **LIST**, **LIST PC**, **PC**, **PE**

LIST PLC

Function List the contents of the specified PLC program.

Scope Global

Syntax **LIST PLC{constant} [,[{start}]] [,[{length}]]**

where:

- **{constant}** is an integer from 0 to 31 representing the number of the PLC program
- the optional **{start}** parameter is an integer constant specifying the distance from the start of the buffer (in words of memory) to begin the listing (the execution point is the default);
- the optional **{length}** parameter (after a comma) is an integer constant specifying the number of words of the buffer to be sent to the host (to the end of the buffer is the default)

Remarks This command causes PMAC to report the contents of the specified uncompiled PLC program buffer to the host. The contents are reported in ASCII text form. If I9 is 0 or 2, the contents are reported in short form (e.g. ENDW). If I9 is 1 or 3, the contents are reported in long form (e.g. ENDWHILE).

If neither **{start}** nor **{length}** is specified, the entire contents of the buffer will be reported. If **{start}** is specified, the reporting will begin **{start}** words from the beginning of the buffer. If **{length}** is specified, the reporting will continue for **{length}** words from the starting point.

If the first comma is present, but no start point is specified, the listing will start from the next line to be executed in the PLC program. Because PMAC can only execute this command between PLC scans, this line will be the first to execute in the next scan. If the second comma is present, but no length is specified, the listing will continue to the end of the

program.

If either **{start}**, **{length}**, or both, or just the comma, is included in the command, the listing of the program will include the buffer address offsets with each line.

PLCs 0-15 can be protected by password. If the PLC is protected by password, and the proper password has not been given, PMAC will reject this command (reporting an ERR002 if I6=1 or 3).

Example

```
LIST PLC 5
P1=0
WHILE (P1<1000)
P1=P1+1
ENDWHILE
RETURN
LIST PLC 5,0
0:P1=0
1:WHILE(P1<1000)
3:P1=P1+1
6:ENDWHILE
7:RETURN
LIST PLC 5,,1
1:WHILE(P1<1000)
LIST PLC 5,,
1:WHILE(P1<1000)
3:P1=P1+1
6:ENDWHILE
7:RETURN
```

See Also PLC Program Features
I-variables I3, I4, I9
On-line commands **LIST**, **LIST PROG**, **PASSWORD={string}**
Program Command Specification

LIST PROGRAM

Function List the contents of the specified motion program.

Scope Global

Syntax

```
LIST PROGRAM {constant} [,[{start}]] [,[{length}]]
LIST PROG {constant} [,[{start}]] [,[{length}]]
```

where:

- **{constant}** is an integer from 1 to 32767 specifying the number of the motion program
- the optional **{start}** parameter is an integer constant specifying the distance from the start of the buffer (in words of memory) to begin the listing (0 is the default);
- the optional **{length}** parameter (after a comma) is an integer constant specifying the number of words of the buffer to be sent to the host (to the end of the buffer is the default)

Remarks This command causes PMAC to report the contents of the specified fixed motion program buffer (PROG) to the host. The contents are reported in ASCII text form. If I9 is 0 or 2, the contents are reported in short form (e.g. LIN). If I9 is 1 or 3, the contents are reported in long form (e.g. LINEAR).

If neither **{start}** nor **{length}** is specified, the entire contents of the buffer will be reported. If **{start}** is specified, the reporting will begin **{start}** words from the

beginning of the buffer. If **{length}** is specified, the reporting will continue for **{length}** words from the starting point.

If either **{start}**, **{length}**, or both, or just the comma, is included in the command, the listing of the program will include the buffer address offsets with each line. Having a listing with these offsets can be useful in conjunction with later use of the **PC** (Program-Counter) and **LIST PC** commands.

If the motion program requested by this command does not exist in PMAC, PMAC will reject this command (reporting an ERR003 if I6=1 or 3).

PROGs 1000-32767 can be protected by password. If the PROG is protected by password, and the proper password has not been given, PMAC will reject this command (reporting an ERR002 if I6=1 or 3).

Example

```
LIST PROG 9                ; Request listing of all of motion program 9
LINEAR .....               ; PMAC responds
F10
X10Y10
X0Y0
RETURN
LIST PROG 9,               ; Request listing of program w/ address offsets
0:LINEAR
1:F10
2:X10Y10 ...               ; Note that a 2-axis command takes 2 addresses
4:X0Y0
6:RETURN
LIST PROG 9,4              ; Request listing starting at address 4
4:X0Y0
6:RETURN
LIST PROG 9,2,4            ; Request listing starting at 2, 4 words long
2:X10Y10
4:X0Y0
LIST PROG 9,,2             ; Request listing starting at top, 2 words long
0:LINEAR
1:F10
```

See Also

Writing a Motion Program
I-variables I3, I4, I9
On-line commands **LIST**, **PC**, **LIST PC**., **PASSWORD={string}**.
Program Command Specification

LIST ROTARY

Function List contents of addressed coordinate system's rotary program buffer

Scope Coordinate-system specific

Syntax

```
LIST ROTARY [{start}] [,{length}]
LIST ROT [{start}] [,{length}]
```

where:

- the optional **{start}** parameter is an integer constant specifying the distance from the start of the buffer (in words of memory) to begin the listing (0 is the default);
- the optional **{length}** parameter (after a comma) is an integer constant specifying the number of words of the buffer to be sent to the host (to the end of the buffer is the

default)

Remarks This command causes PMAC to report the contents of the rotary motion program buffer for the addressed coordinate system to the host. The contents are reported in ASCII text form. If I9 is 0 or 2, the contents are reported in short form (e.g. *LIN*). If I9 is 1 or 3, the contents are reported in long form (e.g. *LINEAR*).

If neither **{start}** nor **{length}** is specified, the entire contents of the buffer will be reported. If **{start}** is specified, the reporting will begin **{start}** words from the beginning of the buffer. If **{length}** is specified, the reporting will continue for **{length}** words from the starting point.

If either **{start}**, **{length}**, or both, or just the comma, is included in the command, the listing of the program will include the buffer address offsets with each line. Having a listing with these offsets can be useful in conjunction with later use of the **PC** (Program-Counter) and **LIST PC** commands.

If the loading of the rotary buffer has caused the buffer to "wrap around" and re-use the beginning of the buffer, the listing will start relative to this new top of the buffer (even if there are previously loaded, and still unexecuted, lines at the bottom of the buffer).

See Also Writing a Motion Program
I-variables I3, I4, I9
On-line commands **LIST**, **PC**, **LIST PE**,
Program Command Specification

LIST TCOMP

Function List contents of addressed motor's torque compensation table

Scope Motor specific

Syntax **LIST TCOMP**

Remarks This command causes PMAC to report to the host the contents of the torque compensation table belonging to the addressed motor. The values are reported in decimal ASCII form, multiple values to a line, with individual values separated by spaces.

The **LIST TCOMP DEF** command should be used to report the header information for this table.

If there is no table for the addressed motor, PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Example
```
LIST TCOMP...                              ; Request contents of backlash comp table
9 17 -3 6 35 87 65 24 18 -9 -16 -34        ; PMAC responds
-7 12 -3 -8 32 44 16 0 -20 -5 0            ; Continued response
```

See Also Backlash Compensation Tables (Setting Up a Motor)
On-line commands **DEFINE TCOMP**, **DELETE TCOMP**, **LIST TCOMP DEF**

LIST TCOMP DEF

Function List definition of addressed motor's torque compensation table

Scope Motor specific

Syntax **LIST TCOMP DEF**

Remarks This command causes PMAC to report to the host the definition of the backlash compensation table that belongs to the addressed motor. The definition reported consists of the two items established in the **DEFINE TCOMP** command that set up the motor:

1. The number of entries in the table;
2. The span of the table in counts of the motor.

If there is no table for the addressed motor, PMAC will reject the command (reporting ERR003 if I6=1 or 3).

Example

```
LIST TCOMP DEF          ; Request def of addressed motor backlash comp table
100,100000 .            ; Turbo PMAC responds; 100 entries in table,
                        ; span is 100,000 counts
```

See Also Backlash Compensation Tables (Setting Up a Motor)
On-line commands **DEFINE TCOMP**, **DELETE TCOMP**, **LIST TCOMP**

M{constant}

Function Report the current M-variable value(s).

Scope Global

Syntax **M{constant}[..{constant}]**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;

Remarks This command causes PMAC to report the current value of the specified M-variable or range of M-variables. It does not cause PMAC to report the definition (address) of the M-variable(s); that is done with the **M{constant}->** command.

> ***Note:***
>
> If a motion program buffer (including a rotary buffer) is open when this command is sent to PMAC it will be entered into the buffer for later execution, to be interpreted as an M-code subroutine call.

Example

```
M0 .....                ; Host asks for value
3548976 .....           ; PMAC's response
M165
5.75
M1..3
1
0
1
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}={expression}**, **M{constant}->**
Program commands **M{constant}**, **M{constant}={expression}**

M{constant}={expression}

Function Assign value to M-variable(s).

Scope Global

Syntax **M{constant}[..{constant}]={expression}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{expression}** contains the value to be given to the specified M-variable(s)

Remarks This command assigns the value on the right side of the equals sign to the specified M-variable(s). It does not assign a definition (address) to the M-variable(s); that is done with the **M{constant}->{definition** command.

If a motion or PLC program buffer is open when this command is sent to PMAC, it will be entered into the buffer for later execution.

Example
```
M1=1
M9=M9 & $20
M102=-16384
M1..8=0
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->{definition}**
Program commands **M{constant}**, **M{constant}={expression}**

M{constant}->

Function Report current M-variable definition(s)

Scope Global

Syntax **M{constant}[..{constant}]->**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;

> ***Note:***
>
> No spaces are permitted between the M-variable name and the "arrow" double character in this command.

Remarks This command causes PMAC to report the definition (address) of the specified M-variable or range of M-variables. It does not cause PMAC to report the *value* of the M-variable{s); that is done with the **M{constant}** command.

When I9 is 0 or 2, only the definition itself (e.g. *Y:$FFC2,0*) is returned. When I9 is 1or 3, the entire definition statement (e.g. *M11->Y:$FFC2,0*) is returned.

Example
```
M1->.....                ; Host requests definition
Y:$FFC2,8.....           ; PMAC's response
M101..103->
X:$C001,24,S
Y:$C003,8,16,S
X:$C003,24,S
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->{definition}**, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant} ->*

Function Self-Referenced M-Variable Definition

Scope Global

Syntax **M{constant}[..{constant}]->***
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;

- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;

> ***Note:***
>
> Spaces are not permitted between the M-variable name and the arrow double character in this command.

Remarks This command causes PMAC to reference the specified M-variable or range of M-variables to its own definition word. If you just wish to use an M-variable as a flag, status bit, counter, or other simple variable, there is no need to find an open area of memory, because it is possible to use some of the definition space to hold the value. Simply define this form of the M-variable and you can use this M-variable much as you would a P-variable, except it only takes integer values in the range -1,048,576 to +1,048,575 (-2^{20} to $+2^{20}-1$).

When the definition is made, the value is automatically set to 0.

This command is also useful to "erase" an existing M-variable definition.

Example
```
M100->*
M20..39->*
M0..1023->*.          ; This erases all existing M-variable definitions
.....                 ; It is a good idea to use this before loading new ones
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**, **M{constant}->{definition}**, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->D:{address}

Function Long Fixed-Point M-Variable Definition

Scope Global

Syntax **M{constant}[..{constant}]->D[:]{address}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{address}** is an integer constant from 0 to 65,535 ($0 to $FFFF if specified in hex).

> ***Note:***
>
> No spaces are permitted between the M-variable name and the arrow double character in this command.

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to a 48-bit double word (both X and Y memory; X more significant) at the specified location in PMAC's address space. The data is interpreted as a fixed-point signed (two's complement) integer.

The definition consists of the letter **D**, an optional colon (**:**), and the word address.

Memory locations for which this format is useful are labeled with **D:** in the memory map.

Example
```
M161->D:$0028         ; Motor 1 desired position register specified in hex
```

```
M161->D40                ; Motor 1 desired position register specified in decimal
M162->D$2C               ; Motor 1 actual position register specified in hex
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**,
M{constant}={expression}
Program command **M{constant}={expression}**

M{constant}->DP:{address}

Function Dual-Ported RAM Fixed-Point M-Variable Definition

Scope Global

Syntax **M{constant}[..{constant}]->DP[:]{address}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{address}** is an integer constant from 0 to 65,535 ($0 to $FFFF if specified in hex).

> ***Note:***
>
> Spaces are not permitted between the M-variable name and the arrow double character in this command.

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to point to 32 bits of data in the low 16 bits of both X and Y memory at the specified location in PMAC's address space. The data is interpreted as a fixed-point signed (two's complement) integer.

The definition consists of the letters **DP**, an optional colon (**:**), and the word address.

This format is only useful for dual-ported RAM locations $D000 to $DFFF (Option 2 is required). With this format, the host can read or write to the corresponding location with a standard 32-bit integer data format. The data in the X word is the most significant word, which means on the host side the most significant word is in the higher of two consecutive addresses (standard Intel format).

Example

```
M150->DP:$D200
M250->DP$D201
```

See Also M-Variables (Computational Features)
Dual-Ported RAM (Writing a Host Communications Program)
On-line commands **M{constant}**, **M{constant}->**,
.....M{constant}->F:{address}, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->F:{address}

Function Dual-Ported RAM Floating-Point M-Variable Definition

Scope Global

Syntax **M{constant}[..{constant}]->F[:]{address}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;

- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{address}** is an integer constant from 0 to 65,535 ($0 to $FFFF if specified in hex).

> ***Note:***
>
> No spaces are permitted between the M-variable name and the arrow double character in this command.

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to point to 32 bits of data in the low 16 bits of both X and Y memory at the specified location in PMAC's address space. The data is interpreted as a floating-point value with the IEEE single-precision (32-bit) format.

The definition consists of the letter **F**, an optional colon (**:**), and the word address.

This format is only useful for dual-ported RAM locations $D000 to $DFFF (Option 2 required). With this format, the host can read or write to the corresponding location with the standard IEEE 32-bit floating-point data format.

The IEEE 32-bit floating point format has the sign bit in bit 31 (MSB); the biased exponent in bits 30 to 23 (the exponent is this value minus 127), and the fraction in bits 22 to 0 (there is an implied 1 added to the fraction in the mantissa). The words are arranged in the standard Intel format.

Example
```
M155->F:$D401
M255->F$D402
```

See Also M-Variables (Computational Features)
Dual-Ported RAM (Writing a Host Communications Program)
On-line commands **M{constant}**, **M{constant}->**,
.....M{constant}->DP:{address}, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->L:{address}

Function Long Word Floating-Point M-Variable Definition

Scope Scope

Syntax **M{constant}[..{constant}]->L[:]{address}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{address}** is an integer constant from 0 to 65,535 ($0 to $FFFF if specified in hex).

> ***Note:***
>
> No spaces are permitted between the M-variable name and the arrow double character in this command.

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to point to a long word (48 bits) of data – both X and Y memory – at the specified location in PMAC's address space. The data is interpreted as a floating-point value with PMAC's own 48-bit floating-point format.

The definition consists of the letter **L**, an optional colon (**:**), and the word address.

Memory locations for which this format is useful are labeled with 'L:' in the memory map.

Example
```
M165->L:$081F
M265->L$0820
M265->L2080
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**,
.....M{constant}->D:{address}, **M{constant}={expression}**
Program command **M{constant}={expression}**

M{constant}->TWB:{multiplex address}

Function Binary Thumbwheel-Multiplexer Definition

Scope Global

Syntax **M{constant}[..{constant}]->TWB[:]{multiplex address},{offset},{size},{format}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{multiplex address}** is an integer constant in the range 0 to 255, representing the byte address in the multiplexing scheme on the thumbwheel port of the least significant bit to be used in the M-variable(s);
- **{offset}** is an integer constant from 0 to 7, representing which bit of this byte is the least significant bit to be used in the M-variable;
- **{size}** is an integer constant from 1 to 32, representing the number of consecutive bits to be used in the M-variable(s);
- **{format}** (optional) is either U for unsigned, or S for signed (two's complement). If no format is specified, U (unsigned) is assumed

> ***Note:***
>
> No spaces are permitted between the M-variable name and the "arrow" double character in this command.

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to a consecutive of input bits multiplexed on the thumbwheel port with Accessory 18 or compatible hardware.

Example
```
M0->TWB:0,0,1
M1->TWB:0,1,1
M10->TWB:3,4,4,U
M745->TWB:4,0,16,S
M872->TWB:0,4,1
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**,
.....M{constant}->TWD:{address}
Thumbwheel Multiplexer Board (ACC-18) Manual

M{constant}->TWD:{address}

Function BCD Thumbwheel-Multiplexer M-Variable Definition

Scope Global

Syntax **M{constant}[..{constant}]->TWD[:]{multiplex address},{offset},{size}[.{dp}],{format}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{multiplex address}** is an integer constant in the range 0 to 255, representing the address in the multiplexing scheme on the thumbwheel port of the most significant digit (lowest address) to be used in the M-variable(s);
- **{offset}** is 0 or 4, representing whether the most significant digit is in the low nibble (left digit of pair) or high nibble (right digit of pair) of the pair of digits at **{multiplex address}**, respectively;
- **{size}** is an integer constant from 1 to 12, representing the number of digits to be used in the M-variable(s);
- **{dp}** (optional) is an integer constant from 0 to 8, representing the number of these digits to be interpreted as being to the right of the decimal point;
- **{format}** (optional) is either U for unsigned, or S for signed. If it is signed, the least significant bit of the most significant digit is taken as the sign bit (the rest of the most significant digit is ignored). If no format is specified, U (unsigned) is assumed.

> ***Note:***
>
> No spaces are permitted between the M-variable name and the arrow double character in this command.

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to point to a set of binary-coded-decimal digits multiplexed on the thumbwheel port with Accessory 18 or compatible hardware.

Thumbwheel-multiplexer M-variables are read-only, floating-point variables. Once defined, they are to be used in expressions, and queried. Each time one is used in an expression, the proper addresses on the multiplexer board(s) are read.

Example **M100->TWD:4,0,8.3,U** means the most significant digit is at multiplex address 4, low nibble (left digit); there are 8 digits, 3 of which are fractional; and it is always interpreted as a positive value. This corresponds to eight thumbwheel digits along the bottom row of the lowest-addressed thumbwheel board, with the decimal point 3 digits in from the right.

M99->TWD:0,0,1,U means that a single digit is used, at multiplex address 0, low nibble (left digit). This corresponds to the upper left thumbwheel on the lowest-addressed thumbwheel board.

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**, **M{constant}->TWB:{address}**
Thumbwheel Multiplexer Board (ACC-18) Manual

M{constant}->TWR:{address},{offset}

Function Resolver Thumbwheel-Multiplexer M-Variable Definition

Scope Global

Syntax **M{constant}[..{constant}]->TWR[:]{multiplex address}, {offset}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{multiplex address}** is an integer constant, divisible by 2, in the range 0 to 254, representing the address in the multiplexing scheme of the ACC-8D Option 7 resolver-to-digital converter board on the thumbwheel multiplexer port, as determined by the DIP switch settings on the board
- **{offset}** is an integer constant from 0 to 7, representing the location of the device at the specified multiplexer address, as determined by in the buffer on the ACC-8D Option 7 and the actual pins to which the device was wired.

Note:

No spaces are permitted between the M-variable name and the arrow double character in this command

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to point to a 12-bit word from a resolver-to-digital (R/D) converter or similar device serially multiplexed on the "thumbwheel" port on an ACC-8D Option 7 or compatible board.

The address on the multiplex port specified here must match the address set by the DIP switches on board the ACC-8D Opt-7. The ACC-8D Opt-7 manual contains a table listing all of the possibilities.

One of the DIP switches on the ACC-8D Opt-7 board determines whether the R/D converters on board have offset values of 0 to 3 or 4 to 7. The **{offset}** specifier must match this DIP switch setting and the number of the R/D device on the board.

This is a read-only M-variable format. Use of this variable in an on-line query command or a program statement will cause PMAC to clock in 12 bits of unsigned data (range 0 to 4095) from the specified device through the multiplexer port.

Note:

It is not necessary to use an M-variable to access an R/D converter for actual servo or phasing feedback purposes. I-variables (Ix10, Ix81, I8x, I9x) are used for that purpose. However, even if this is your only use of the R/D converter, it is usually desirable to assign M-variables to the R/D converters for set-up and diagnostic purposes.

Example
```
M100->TWR:0,0
M99->TWR:4,5
```

M{constant}->TWS:{address}

Function Serial Thumbwheel-Multiplexer M-Variable Definition

Scope Global

Syntax **M{constant}[..{constant}]->TWS[:]{multiplex address}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{multiplex address}** is an integer constant, divisible by 4, in the range 0 to 124, representing the address in the multiplexing scheme of the first of four bytes in the 32-bit

input or output word. Adding 1 to the **{multiplex address}** designates it as a read-only variable and adding 2 designates it as a write-only variable.

> ***Note:***
>
> No spaces are permitted between the M-variable name and the arrow double character in this command.

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to point to a 32-bit word of input or output serially multiplexed on the "thumbwheel" port on an Accessory 34x board.

> ***Note:***
>
> The individual bits of the thumbwheel port on an Accessory 34x board can not be directly assigned to an M-variable. Only 32-bit words (ports) of input or output can be accessed.

The address on the multiplex port specified here must match the address set by the DIP switch on board the ACC-34x. The ACC-34x manual contains a table listing all of the possibilities.

The entire word must either be all input or all output. On power-up/reset, all ACC-34x words are software-configured as inputs (if the hardware is configured for outputs, all outputs will be OFF – pulled up to the supply voltage). Any subsequent write operation to an I/O word on the port with one of these M-variables automatically makes the entire word an output word, with individual bits ON or OFF, as determined by the value written to the word.

Any subsequent read operation of a word that has been set up for output configures, or tries to configure, the entire word into an input word, which turns any hardware outputs OFF. Therefore, it is important that the following rules be observed when working with these M-variables:

Never use this M-variable form to write to a word that is set up for inputs.
Never use this M-variable form to read from a word that is set up for outputs.

Because both reads and writes are enabled when a TWS type M-variable is used to point directly to the base address of an accessory 34x port (e.g. **M300->TWS:40**) their use is very strongly discouraged. Reads and writes are enabled when the least significant and the next least significant addresses bits are both zero (e.g. decimal 40 = 01000000 in binary).

In this situation, any accidental read of an output port (say via the Executive programs watch window) will cause all the output transistors to be turned off (outputs pulled to the supply voltage)! Alternatively, writing to an input port, will automatically reconfigure it to an output port! It is therefore safer and more predictable when bits 0 & 1 of the M-variable definition are intentionally used to disable either the read function or the write function. Setting one of these bits gives the read-only or write-only form of the TWS M-variable.

An M-variable pointing to an input port is defined as read-only by setting the **{multiplex address}** to a legal byte number (from column 2 of Table 1 of the Acc 34x manual) ***plus 1***. Any attempt to write to a TWS type M-variable defined in this manner (with bit zero of its address set to 1) is automatically prevented by PMAC firmware. For an output port, the **{multiplex address}** should be a legal byte number (from column 2 of Table 1 of the

Acc 34x manual) plus *2*. Any attempt to read a TWS type M-variable defined in this manner (with bit one of its address set to 1) returns zero and the actual read is prevented by PMAC firmware.

Because you can not directly access the individual bits of the "thumbwheel port on an Accessory 34x board and because of the relatively long time it takes to clock the data in or out of PMAC (A 32-bit Read or a 32-bit Write to an individual port takes approximately 64 microseconds of time in the PMAC's background time slot) it is best to keep an "image" of each M-variable of this type in internal memory. The image variable would preferably be a 32-bit or 48-bit fixed point M-variable, but it could also be a 48-bit floating point P or Q variable.

The best procedure for using TWS M-variables in a program is as follows. The input word (TWS M-variable) should be copied into its image variable at the beginning of a sequence of operations. The operations can then be done on the image variable without requiring PMAC to actually read or write to the I/O port for each operation. The output word is first "assembled" into its image variable, and then copied to the actual output word once at the end of a sequence of operations. This procedure will allow the most efficient and flexible use of TWS M-variables.

This type of variable can only be used in background tasks (PLCs and PLCCs 1-31). They cannot be used in foreground tasks (motion programs and PLC and PLCC 0).

Example To address Port B of board #1 as an output using M101, use the following definition. This addressing format is not recommended because accidental reads of the port are not protected against (Consider using the write-only format).

```
M100->TWS:4          ;Port B  (BIO 0-31) of an  ACC-34x with SW1 switches all on
                     ;not assigned for write-only
```

To address Port B as above using the write-only addressing format, use the following definition:

```
M100->TWS:6          ;Port B  (BIO 0-31) of an  ACC-34x with SW1 switches all ON
                     ;assigned for write-only (6=4+2)
```

To address Port A of board #1 as an input using M99, use the following definition. This addressing format is not recommended because accidental writing to the port is not protected against (Consider using the read-only format).

```
M99->TWS:0           ;Port A (AIO 0-31) of an ACC-34x with SW1 switches all on
                     ; not assigned for read-only
```

To address Port A as above using the read-only addressing format, use the following definition:

```
M100->TWS:1          ;Port A (AIO 0-31) of an ACC-34x with SW1 switches all on
                     ;assigned for read-only (1=0+1)
```

Yet another example: to address Port A of board #6 as an input using M300, we would use the following definition:

```
M300->TWS:41         ;Port A (AIO 0-31) of an ACC-34x with SW1 switches
                     ; ON, ON, OFF, ON, ON assigned for read-only (41=40+1)
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**, **M{constant}->TWR:{address}**
Serial I/O Multiplexer Board (ACC-34) Manual

M{constant}->X/Y:{address}

Function Short Word M-Variable Definition

Scope Global

Syntax
```
M{constant}[..{constant}]->
      X[:]{address},{offset}[,{width}[,{format}]]
M{constant}[..{constant}]->
      Y[:]{address},{offset}[,{width}[,{format}]]
```
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the M-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{address}** is an integer constant from 0 to 65,535 ($0 to $FFFF if specified in hex);
- **{offset}** is an integer constant from 0 to 23, representing the starting (least significant) bit of the word to be used in the M-variable(s), or 24 to specify the use of all 24 bits;
- **{width}** (optional) is an integer constant from the set {1, 4, 8, 12, 16, 20, 24}, representing the number of bits from the word to be used in the M-variable(s); if **{width}** is not specifed, a value of 1 is assumed;
- **{format}** (optional) is a letter from the set [U, S, D, C], specifying how PMAC is to interpret this value: (U=Unsigned integer, S=Signed integer, D=Binary-coded Decimal, C=Complementary binary-coded decimal); if **{format}** is not specified, U is assumed.

> ***Note:***
>
> No spaces are permitted between the M-variable name and the arrow double character in this command.

Remarks This command causes PMAC to define the specified M-variable or range of M-variables to point to a location in one of the two halves (X or Y) of PMAC's data memory. In this form, the variable can have a width of 1 to 24 bits and can be decoded several different ways, so the bit offset, bit width, and decoding format must be specified (the bit width and decoding format do have defaults.

The definition consists of the letter **X** or **Y**, an optional colon (**:**), the word address, the starting bit number (offset), an optional bit width number, and an option format-specifying letter.

Legal values for bit width and bit offset are inter-related. The table below shows the possible values of **{width}**, and the corresponding legal values of **{offset}** for each setting of **{width}**.

{width}	{offset}
1	0 – 23
4	0,4,8,12,16,20
8	0,4,8,12,16
12	0,4,8,12
16	0,4,8
20	0,4

24 0

The format is irrelevant for 1-bit M-variables, and should not be included for them. If no format is specified, 'U' is assumed.

Example

```
; Machine Output 1
M1->Y:$FFC2,8,1          ; 1-bit (full spec.)
M1->Y$FFC2,8             ; 1-bit (short spec.)

; Encoder 1 Capture/Compare Register
M103->X:$C003,0,24,U     ; 24-bit (full spec.)
M103->X$C003,24          ; 24-bit (short spec.)

; DAC 1 Output Register
M102->Y:$C003,8,16,S     ; 16-bit value
M102->Y49155,8,16,S      ; same, decimal address
```

See Also M-Variables (Computational Features)
On-line commands **M{constant}**, **M{constant}->**, **M{constant}->D:{address}**, **M{constant}={expression}**
Program command **M{constant}={expression}**

MACROAUX

Function Report or write MACRO auxiliary parameter value

Scope Global

Syntax

```
MACROAUX{NodeNum}{ParamNum}[={constant}]
MX{NodeNum}{ParamNum}[={constant}]
```

where:

- **{NodeNum}** is an integer constant from 0 to 15 specifying the slave number of the node
- **{ParamNum}** is an integer constant from 0 to 65535 specifying the auxiliary parameter number for this node (2 to 254 required for a write operation)
- **{constant}** is an integer constant from -32768 to +32767 representing the value to be written to the specified parameter

Remarks This command permits PMAC to read or write auxiliary register values from slave nodes across the MACRO ring. The command must specify the node number of the slave node, the auxiliary parameter number at this node, and if a write command, the value to write into the register.

If used as a read command (no '**={constant}**' in the command), PMAC will report the value of the specified parameter back to the host as ASCII text, just as if the value of one of its own parameters had been requested.

Only one auxiliary access (read or write) of a single node can be done on one command line.

In order to access the auxiliary registers of a MACRO node *n*, bit *n* of I1000 must be set to 1.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example

```
MACROAUX1,24=2000        ; Set Node 1 Parameter 24 to 2000
MACROAUX1,24             ; Request value of Node 1 Parameter 24
```

```
2000                    ; PMAC reports value
```

See Also On-line commands **MACROAUXREAD, MACROAUXWRITE**
PLC Program commands **MACROAUXREAD, MACROAUXWRITE**

MACROAUXREAD

Function Read MACRO auxiliary parameter value

Scope Global

Syntax **MACROAUXREAD{NodeNum}{ParamNum}{Variable}**
MXR{NodeNum}{ParamNum}{Variable}
where:

- **{NodeNum}** is an integer constant from 0 to 15 specifying the slave number of the node
- **{ParamNum}** is an integer constant from 0 to 65535 specifying the auxiliary parameter number for this node
- **{Variable}** is the name of the PMAC variable (I, P, Q, or M) into which the parameter value is to be copied

Remarks This command permits PMAC to read auxiliary register values from slave nodes across the MACRO ring. The command must specify the node number of the slave node, the auxiliary parameter number at this node, and the name of the PMAC variable to receive the value.

Only one auxiliary access (read or write) of a single node can be done on one command line.

In order to access the auxiliary registers of a MACRO node *n*, bit *n* of I1000 must be set to 1.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example

```
MACROAUXREAD1,24,P1          ; Read Node 1 Parameter 24 into P1
MXR5,128,M100                ; Read Node 5 Parameter 128 into M100
```

See Also On-line commands **MACROAUX, MACROAUXWRITE**
PLC Program commands **MACROAUXREAD, MACROAUXWRITE**

MACROAUXWRITE

Function Write MACRO auxiliary parameter value

Scope Global

Syntax **`MACROAUXWRITE{NodeNum}{ParamNum}{Variable}`**
`MXW{NodeNum}{ParamNum}{Variable}`
where:

- **`{NodeNum}`** is an integer constant from 0 to 15 specifying the slave number of the node
- **`{ParamNum}`** is an integer constant from 2 to 253 specifying the auxiliary parameter number for this node
- **`{Variable}`** is the name of the PMAC variable (I, P, Q, or M) from which the parameter value is to be copied

Remarks This command permits PMAC to write auxiliary register values to slave nodes across the MACRO ring. The command must specify the node number of the slave node, the auxiliary parameter number at this node, and the name of the PMAC variable from which the value comes.

Only one auxiliary access (read or write) of a single node can be done on one command line.

In order to access the auxiliary registers of a MACRO node *n*, bit *n* of I1000 must be set to 1.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example

```
MACROAUXWRITE1,24,P1      ; Write value of P1 to Node 1 Parameter 24
MXW5,128,M100             ; Write value of M100 to Node 5 Parameter 128
```

See Also On-line commands **`MACROAUX, MACROAUXREAD`**
PLC Program commands **`MACROAUXREAD, MACROAUXWRITE`**

MACROSLV{command} {node#}

Function Send command to Type 1 MACRO slave

Scope Global

Syntax **`MACROSLAVE{command}{node #}`**
`MS{command}{node #}`
where:

- **`{command}`** is one of the following text strings:
  - **`$$$`** normal station reset
  - **`$$$***`** station reset and re-initialize
  - **`CLRF`** station fault clear
  - **`CONFIG`** report station configuration value
  - **`DATE`** report station firmware date
  - **`SAVE`** save station setup
  - **`VER`** report station firmware version
  - **`{node #}`** is a constant in the range 0 to 15 representing the number of the node on the PMAC matching the slave node to be accessed

Remarks This command causes PMAC to issue the specified command to a MACRO slave station using the Type 1 auxiliary master-to-slave protocol. If **{node #}** is set to 15, the action automatically applies to all slave stations commanded from the PMAC.

The **MS CONFIG** command allows the user to set and report a user-specified configuration value. This provides any easy way for the user to see if the MACRO station has already been configured to the user's specifications. The factory default configuration value is 0. It is recommended that after the user finishes the software configuration of the station, a special number be given to the configuration value with the **MS CONFIG{node #}={constant}** command. This number will be saved to the non-volatile memory with the **MS SAVE** command.

Subsequently, when the system is powered up, the station can be polled with the **MS CONFIG {node #}** command. If the expected value is returned, the station can be assumed to have the proper software setup. If the expected value is not returned (for instance, when a replacement station has just been installed) then the setup will have to be transmitted to the station.

In order for the PMAC to be able to execute this command, the following conditions must be true:

- The PMAC must be set up as a master or the synchronizing ring master (I995= $xx90 or $xx30);
- The node 15 auxiliary register copy function must be disabled (I1000 bit 15 = 0);
- Node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I1003 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example

```
MS $$$0                 ; Resets MACRO station which has active node 0
MS $$$***4              ; Reinitializes MACRO station which has active node 4
MS CLRF8                ; Clears fault on Node 8 of MACRO station
MS CONFIG12             ; Causes MACRO station to report its configuration #
37                      ; PMAC reports MACRO station configuration # to host
MS CONFIG12=37          ; Sets MACRO station configuration number
MS DATE 0               ; Causes MACRO station to report its firmware date
08/12/1999              ; PMAC reports MACRO station firmware date to host
MS SAVE 4               ; Causes MACRO station to save setup variables
MS VER 8                ; Causes MACRO station to report its firmware version
1.104                   ; PMAC reports MACRO station firmware version to host
```

MACROSLV{node#},{slave variable}

Function Report Type 1 MACRO auxiliary parameter value

Scope Global

Syntax

```
MACROSLAVE{node #},{slave variable}
MS{node #},{slave variable}
```

where:

- **{node #}** is a constant in the range 0 to 15 representing the number of the node on the PMAC matching the slave node to be accessed
- **{slave variable}** is the name of the variable on the slave station whose value is to be reported

Remarks This command causes PMAC to query the MACRO slave station at the specified node number using the MACRO Type 1 master-to-slave auxiliary protocol, and report back the value of the specified slave station variable to the host computer.

In order for the PMAC to be able to execute this command, the following conditions must be true:

- The PMAC must be set up as a master or the synchronizing ring master (I995= $xx90 or $xx30);
- The node 15 auxiliary register copy function must be disabled (I1000 bit 15 = 0);
- Node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I1003 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example

```
MS0,MI910          ; Causes slave to report value of Node 0 variable MI910
7                  ; PMAC reports this value back to host
MS1,MI997          ; Causes slave to report value global variable MI997
                   ; PMAC reports this value back to host
```

MACROSLV{node#},{slave variable}={constant}

Function Set Type 1 MACRO auxiliary parameter value

Scope Global

Syntax

```
MACROSLAVE{node #},{slave variable}={constant}
MS{node #},{slave variable}={constant}
```

where:

- **{node #}** is a constant in the range 0 to 15 representing the number of the node on the PMAC matching the slave node to be accessed
- **{slave variable}** is the name of the MI-variable or C-command on the slave station whose value is to be set;
- **{constant}** is a number representing the value to be written to the specified MI-variable

Remarks This command causes PMAC to write the specified constant value to the variable of the MACRO slave station at the specified node number using the MACRO Type 1 master-to-slave auxiliary protocol.

In order for the PMAC to be able to execute this command, the following conditions must be true:

- The PMAC must be set up as a master or the synchronizing ring master (I995= $xx90 or $xx30);
- The node 15 auxiliary register copy function must be disabled (I1000 bit 15 = 0);
- Node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I1003 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example
```
MS0,MI910=7          ; Causes slave to set value of Node 0 variable MI910 to 7
MS1,MI997=6528       ; Causes slave to set value global variable MI997 to 6528
MS8,C2=0             ; Causes slave at node 8 to reset
```

MACROSLVREAD

Function Read (copy) Type 1 MACRO auxiliary parameter value

Scope Global

Syntax
```
MACROSLVREAD{node #},{slave variable},{PMAC variable}
MSR{node #},{slave variable},{PMAC variable}
```
where:

- **{node #}** is a constant in the range 0 to 15 representing the number of the node on the PMAC matching the slave node to be accessed
- **{slave variable}** is the name of the variable on the slave station whose value is to be reported
- **{PMAC variable}** is the name of the variable on the PMAC into which the value of the slave station variable is to be copied

Remarks This command causes PMAC to copy the value of the specified variable of the MACRO slave station matching the specified node number on the PMAC to the specified PMAC variable, using the MACRO Type 1 master-to-slave auxiliary protocol.

The variable on the PMAC can be any of the I, P, Q, or M-variable on the card.

If this command is issued to the PMAC while a PLC buffer is open, it will be stored in the buffer as a PLC command, not executed as an on-line command.

In order for the PMAC to be able to execute this command, the following conditions must be true:

- The PMAC must be set up as a master or the synchronizing ring master (I995= $xx90 or $xx30);
- The node 15 auxiliary register copy function must be disabled (I1000 bit 15 = 0);
- Node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I1003 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example
```
MSR0,MI910,P1        ; Copies value of slave Node 0 variable MI910 into PMAC variable P1
MSR1,MI997,M10       ; Copies value of slave Node 1 variable MI997 into PMAC variable M10
```

MACROSLVWRITE

Function Write (copy) Type 1 MACRO auxiliary parameter value

Scope Global

Syntax
```
MACROSLVWRITE{node #},{slave variable},{PMAC variable}
MSW{node #},{slave variable},{PMAC variable}
```
where:

- **{node #}** is a constant in the range 0 to 15 representing the number of the node on the PMAC matching the slave node to be accessed
- **{slave variable}** is the name of the MI-variable or C-command on the slave station whose value is to be set;

- **{PMAC variable}** is the name of the variable on the PMAC from which the value of the slave station variable is to be copied

Remarks This command causes PMAC to copy the value of the specified variable on PMAC to the specified variable of the MACRO slave station matching the specified node number on the PMAC, using the MACRO Type 1 master-to-slave auxiliary protocol.

The variable on the PMAC can be any of the I, P, Q, or M-variables on the card.

If this command is issued to the PMAC while a PLC buffer is open, it will be stored in the buffer as a PLC command, not executed as an on-line command.

In order for the PMAC to be able to execute this command, the following conditions must be true:

- The PMAC must be set up as a master or the synchronizing ring master (I995= $xx90 or $xx30);
- The node 15 auxiliary register copy function must be disabled (I1000 bit 15 = 0);
- Node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I1003 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example

```
MSW0,MI910,P35      ; Copies value of PMAC P35 into MACRO station node 0 variable MI910
MSW4,C4,P0          ; Causes MACRO station with active node 4 to execute Command #4,
                    ; saving its setup variable values to non-volatile memory (P0 is a dummy
                    ; variable here)
```

MFLUSH

Function Clear pending synchronous M-variable assignments

Scope Coordinate-system specific

Syntax **MFLUSH**

Remarks This command permits the user to clear synchronous M-variable assignment commands that have been put on the stack for intended execution with a subsequent move (without executing the commands). As an on-line command, it is useful for making sure pending outputs are not executed after a program has been stopped.

Example

```
/                   ; Stop execution of a program
MFLUSH              ; Clear M-variable stack
B1R                 ; Start another program; formerly pending M-variables will not execute
```

See Also Program commands **M{constant}=={expression}**, **M{constant}&={expression}**, **M{constant}|={expression}**, **M{constant}^={expression}**

O{constant}

Function Open loop output

Scope Motor specific

Syntax **O{constant}**
where:

- **{constant}** is a floating-point value representing the magnitude of the output as a percentage of Ix69 for the motor, with a range of +/-100

Remarks This command causes PMAC to put the motor in open-loop mode and force an output of the specified magnitude, expressed as a percentage of the maximum output parameter for the motor (Ix69). This command is commonly used for set-up and diagnostic purposes (for instance, a positive **O** command must cause position to count in the positive direction, or closed-loop control cannot be established), but it can also be used in actual applications.

If the motor is *not* PMAC-commutated, this command will create a DC output voltage on the single DAC for the motor. If the motor is commutated by PMAC, the commutation algorithm is still active, and the specified magnitude of output is apportioned between the two DAC outputs for the motor according to the instantaneous commutation phase angle.

If the value specified is outside the range +/-100, the output will saturate at +/-100% of Ix69.

Closed-loop control for the motor can be re-established with the **J** command. It is a good idea to stop the motor first with an **O0** command if it has been moving in open-loop mode.

To do a variable O-command, define an M-variable to the filter result register (X:$003A, etc.), command an **O0** to the motor to put it in open-loop mode, then assign a variable value to the M-variable. This technique will even work on PMAC-commutated motors.

PMAC will reject this command if the motor is in a coordinate system that is currently running a motion program (reporting ERR001 if I6 is 1 or 3).

Example

```
O50 ..... ; Open-loop output 50% of Ix69 for addressed motor
#2O33.333..... ; Open-loop output 1/3 of Ix69 for Motor 2
O0 ..... ; Open-loop output of zero magnitude
J/ ; Re-establish closed-loop control
```

See Also On-line commands **J/**, **K**
Memory-map registers X:$003A, X:$0076, etc.
Suggested M-variable definitions Mx71.

OPEN BINARY ROTARY

Function Open rotary buffer for entry of binary commands only

Scope Global

Syntax

```
OPEN BINARY ROTARY
OPEN BIN ROT
```

Remarks This command causes PMAC to open all existing rotary motion program buffers (created with the **DEFINE ROTARY** command) for entry of binary-format program commands through the dual-ported RAM only. Subsequent binary-format program commands valid for rotary motion programs that are sent to the appropriate dedicated binary buffer in the DPRAM are copied into the internal rotary program buffer for the appropriate coordinate system.

This effect of this command differs from that of the **OPEN ROTARY** command. After the **OPEN ROTARY** command, ASCII text commands that can be buffered motion program commands are entered into the internal rotary program buffer, as well as binary-format commands (if I57=1). After the **OPEN BINARY ROTARY** command, no ASCII text commands can be entered into the internal rotary buffers. If a text command can be interpreted as an on-line command, it is executed immediately. If it cannot be interpreted as an on-line command, it is rejected with an error.

No other program buffers (PLC, fixed or rotary motion) may be open when the command is sent (PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** command to make sure no other buffers have been left open.

When the rotary buffers are open for binary entry only, bit 6 of Y:$0003 (a new status bit – part of the 11th digit reported in response to the **???** global status query command) is set to 1. However, bits 19 and 18 of this word (part of the 8th digit reported) – Motion Buffer Open and Rotary Buffer Open – are left at 0 to keep ASCII commands out of this buffer.

The binary rotary buffers can be closed for entry with the **CLOSE** command.

See Also I-variable I57
On-line commands **CLOSE**, **OPEN ROTARY**, **???**

OPEN PLC

Function Open a PLC program buffer for entry

Scope Global

Syntax **OPEN PLC {constant}**
where:

- **{constant}** is an integer from 0 to 31 representing the PLC program to be opened

Remarks This command causes PMAC to open the specified PLC program buffer for entry and editing. This permits subsequent program lines that are valid for a PLC to be entered into this buffer. When entry of the program is finished, the **CLOSE** command should be used to prevent further lines from being put in the buffer.

No other program buffers (PLC, fixed or rotary motion) may be open when this command is sent (PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** command to make sure no other buffers have been left open.

PLCs 0-15 can be protected by password. If the PLC is protected by password, and the proper password has not been given, PMAC will reject this command (reporting an ERR002 if I6=1 or 3).

Opening a PLC program buffer automatically disables that PLC program. Other PLC programs and motion programs will keep executing. Closing the PLC program buffer after entry does not re-enable the program. To re-enable the program, the **ENABLE PLC** command must be used, or PMAC must be reset (with a saved value of I5 permitting this PLC program to execute).

Example

```
CLOSE .....            ; Make sure other buffers are closed
DELETE GATHER          ; Make sure memory is free
OPEN PLC 7 ...         ; Open buffer for entry, disabling program
CLEAR .....            ; Erase existing contents
IF (M11=1) ...         ; Enter new version of program...
  ...
CLOSE .....            ; Close buffer at end of program
ENABLE PLC 7           ; Re-enable program
```

See Also PLC Program Features
I-variable I5
On-line commands **CLOSE**, **DELETE GATHER**, **ENABLE PLC**

OPEN PROGRAM

Function Open a fixed motion program buffer for entry

Scope Global

Syntax
```
OPEN PROGRAM {constant}
OPEN PROG {constant}
```
where:

- **{constant}** is an integer from 1 to 32767 representing the motion program to be opened.

Remarks This command causes PMAC to open the specified fixed (non-rotary) motion program buffer for entry or editing. Subsequent program commands valid for motion programs will be entered into this buffer. When entry of the program is finished, the **CLOSE** command should be used to prevent further lines from being put in the buffer.

No other program buffers (PLC, fixed or rotary motion) may be open when this command is sent (PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** command to make sure no other buffers have been left open.

No motion programs may be running in any coordinate system when this command is sent (PMAC will report *ERR001* if I6=1 or 3). As long as a fixed motion program buffer is open, no motion program may be run in any coordinate system (PMAC will report *ERR015* if I6=1 or 3).

PROGs 1000-32767 can be protected by password. If the PROG is protected by password, and the proper password has not been given, PMAC will reject this command (reporting an ERR002 if I6=1 or 3).

After any fixed motion program buffer has been opened, each coordinate system must be commanded to point to a motion program with the **B{constant** command before it can run a motion command (otherwise PMAC will report *ERR015* if I6=1 or 3)

Example
```
CLOSE ..............  ; Make sure other buffers are closed
DELETE GATHER DELETE TRACE  ; Make sure memory is free
OPEN PROG 255  ; Open buffer for entry, disabling program
CLEAR ..............  ; Erase existing contents
X10 Y20 F5 ...  ; Enter new version of program...
   ...
CLOSE ..............  ; Close buffer at end of program
&1B255R  ; Point to this program and run it
```

See Also Writing a Motion Program
On-line commands **CLEAR**, **CLOSE**, **DELETE GATHER**, **DELETE TRACE**
Program Command Specification

OPEN ROTARY

Function Open all existing rotary motion program buffers for entry

Scope Global

Syntax **OPEN ROTARY**
OPEN ROT

Remarks This command causes PMAC to open all existing rotary motion program buffers (created with the **DEFINE ROTARY** command) for entry. Subsequent program commands valid for rotary motion programs are entered into the rotary program buffer of the coordinate system addressed at the time of that command. (Branching and looping commands should not be used in a rotary program buffer.)

No other program buffers (PLC, fixed or rotary motion) may be open when this command is sent (PMAC will report *ERR007* if I6=1 or 3). It is a good idea always to precede an **OPEN** command with a **CLOSE** command to make sure no other buffers have been left open.

The **<CTRL-U>** command performs the same function as **OPEN ROTARY**.

> ***Note:***
>
> The **B0** command that points the coordinate system to the rotary buffer cannot be given while the rotary buffers are open, because PMAC will interpret the command as a B-axis move command.

Example

```
&2 DEFINE ROT 100          ; Create C.S. 2 rotary buffer
&1 DEFINE ROT 100          ; Create C.S. 1 rotary buffer
&1 B0 &2 B0.               ; Point both C.S.s to rotary buffers
OPEN ROT.....              ; Open buffers for entry
&1 X10 Y10 F5              ; Write to C.S. 1's buffer
&2 X30 Y30 F10             ; Write to C.S. 2's buffer
&1R &2R                    ; Start executing both buffers
```

See Also Rotary Motion Programs (Writing a Motion Program)
On-line commands **<CTRL-L>**, **<CTRL-U>**, **CLOSE**, **DEFINE ROT**, **B{constant}**, **R**

P

Function Report motor position

Scope Motor specific

Syntax **P**

Remarks This command causes PMAC to report the present actual position for the addressed motor to the host, scaled in counts, rounded to the nearest tenth of a count.

PMAC reports the value of the actual position register plus the position bias register plus the compensation correction register, and if bit 16 of Ix05 is 1 (handwheel offset mode), minus the master position register.

Example

```
P.....                     ; Request the position of the addressed motor
1995.....                  ; PMAC responds
#1P.....                   ; Request position of Motor 1
-0.5.....                  ; PMAC responds
#2P#4P.....                ; Request positions of Motors 2 and 4
9998.....                  ; PMAC responds with Motor 2 position first
10002                      ; PMAC responds with Motor 4 position next
```

See Also On-line commands **<CTRL-P>**, **F**, **V**
Suggested M-variable definitions Mx62, Mx64, Mx67, Mx69
Memory map registers D:$002B, D:$0813, D:$002D, D:$0046, etc.

P{constant}

Function Report the current P-variable value(s).

Scope Global

Syntax **P{constant}[..{constant}]**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the P-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;

Remarks This command causes PMAC to report the current value of the specified P-variable or range of P-variables.

Example

```
P1 ..................  ; Host asks for value
25 ..................  ; PMAC responds
P1005
3.444444444
P100..102
17.5
-373
0.0005
```

See Also P-Variables (Computational Features)
On-line commands **I{constant}**, **M{constant}**, **Q{constant}**, **P{constant}={expression}**

P{constant}={expression}

Function Assign a value to a P-variable.

Scope Global

Syntax **P{constant}[..{constant}]={expression}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the P-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{expression}** contains the value to be given to the specified P-variable(s)

Remarks This command causes PMAC to set the specified P-variable or range of P-variables equal to the value on the right side of the equals sign.

Example

```
P1=1
P75=P32+P10
P100..199=0
P10=$2000
P832=SIN(3.14159*Q10)
```

See Also P-Variables (Computational Features)
On-line commands **I{constant}={expression}**, **M{constant}={expression}**, **Q{constant}={expression}**, **P{constant}**
Program command **P{constant}={expression}**

PASSWORD={string}

Function Enter/Set Program Password

Scope Global

Syntax **PASSWORD={string}**
where:

- **{string}** is a series of non-control ASCII characters (values from 32 decimal to 255 decimal). The password string is case sensitive.

Remarks This command permits the user to enter the card's password, or once entered properly, to change it. Without a properly entered password, PMAC will not open or list the contents of any motion program numbered 1000 or greater, or of PLC programs 0-15. If asked to do so, it will return an error (ERR002 reported if I6 is set to 1 or 3).

The default password is the null password (which means no password is needed to list the programs). This is how the card is shipped from the factory, and after a **$$$***** re-initialization command. When there is a null password, you are automatically considered to have entered the correct password on power-up/reset.

If you have entered the correct password (which is always the case for the null password), PMAC interprets the **PASSWORD={string}** command as changing the password, and you can change it to anything you want. When the password is changed, it has automatically been matched and the host computer has access to the protected programs.

> ***Note:***
>
> The password does not require quote marks. If you use quote marks when you enter the password string for the first time, you must use them every time you match this password string.

If you have not yet entered the correct password since the latest power-up/reset, PMAC interprets the **PASSWORD={string}** command as an attempt to match the existing password. If the command matches the existing password correctly, PMAC accepts it as a valid command, and the host computer has access to the protected programs until the PMAC is reset or has its power cycled. If the command does not match the existing password correctly, PMAC returns an error (reporting ERR002 if I6=1 or 3), and the host computer does not have access to the protected programs. The host computer is free to attempt to match the existing password.

There is no way to read the current password. If the password is forgotten and access to the protected programs is required, the card must be re-initialized with the **$$$*** command, which clears all program buffers as well as the password. Then the programs must be reloaded, and a new password entered.

Example

```
{Starting from power-up/reset with a null password}
LIST PLC 1                ; Request listing of protected program
P1=P1+1                   ; PMAC responds because there is no password
RETURN
PASSWORD=Bush             ; This sets the password to "Bush"
LIST PLC 1                ; Request listing of protected program
P1=P1+1                   ; PMAC responds because password has been
RETURN                    ; matched by changing it.
$$$                       ; Reset the card
LIST PLC 1                ; Request listing of protected program
ERR002                    ; PMAC rejects because password not entered
```

```
PASSWORD=Reagan          ; Attempt to enter password
ERR002                   ; PMAC rejects as incorrect password
PASSWORD=BUSH            ; Attempt to enter password
ERR002                   ; PMAC rejects as incorrect (wrong case)
PASSWORD=Bush            ; Attempt to enter password
                         ; PMAC accepts as correct password
LIST PLC 1               ; Request listing of protected program
P1=P1+1                  ; PMAC responds because password matched
RETURN
PASSWORD=Clinton         ; This changes password to "Clinton"
LIST PLC 1               ; Request listing of protected program
P1=P1+1                  ; PMAC responds because password has been
RETURN                   ; matched by changing it.
$$$                      ; Reset the card
PASSWORD=Clinton         ; Attempt to enter password
                         ; PMAC accepts as correct password
LIST PLC 1               ; Request listing of protected program
P1=P1+1                  ; PMAC responds because password matched
RETURN
```

See Also On-line commands **LIST**, **LIST PC**, **LIST PE**, **OPEN**

PAUSE PLC

Function Pause specified PLC program(s).

Scope Global

Syntax

```
PAUSE PLC {constant}[,{constant}...]
PAU PLC {constant}[,{constant}...]
PAUSE PLC {constant}[..{constant}]
PAU PLC {constant}[..{constant}]
```

where:

- **{constant}** is an integer from 0 to 31, representing the program number

Remarks This command causes PMAC to stop execution of the specified uncompiled PLC program or programs, with the capability to restart execution at this point (not necessarily at the top) with a **RESUME PLC** command. Execution can also be restarted at the top of the program with the **ENABLE PLC** command.

The on-line **PAUSE PLC** command can only suspend execution of a PLC program either at the end of a scan, which is the end of the program, or at an **ENDWHILE** statement in the program.

PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs.

If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.

Example

```
PAUSE PLC 1
PAU PLC 5
PAU PLC 3,4,7
PAUSE PLC 0..31
```

See Also I-variable I5

On-line commands **DISABLE PLC**, **ENABLE PLC**, **OPEN PLC**, **RESUME PLC**, **LIST PLC**, **<CONTROL-D>**.

Program commands **DISABLE PLC**, **ENABLE PLC**, **PAUSE PLC**, **RESUME PLC**

PC

Function Report Program Counter

Scope Coordinate-system specific

Syntax **PC**

Remarks This command causes PMAC to report the motion program number and address offset of the line in that program that it will next calculate (in the addressed coordinate system). It will also report the program number and address offset of any lines it must **RETURN** to if it is inside a **GOSUB** or **CALL** jump (up to 15 deep).

The number reported after the colon is not a line number; as an address offset, it is the number of words of memory from the top of the program. The **LIST PROGRAM** command, when used with comma delimiters, shows the program or section of the program with address offsets for each line. The **LIST PC** command can show lines of the program with address offsets from the point of calculation.

Because PMAC calculates ahead in a continuous sequence of moves, the **PC** (Program Calculation) command will in general return a program line further down in the program than **PE** will.

If the coordinate system is not pointing to any motion program, PMAC will return an error (ERR003 if I6=1 or 3). Initially the pointing must be done with the **B{constant}** command.

Example

```
PC
P1:0.....                    ; Ready to execute at the top of PROG 1
PC
P76:22.....                  ; Ready to execute at 22nd word of PROG 76
LIST PC
P76:22:X10Y20                ; Program line at 22nd word of PROG 76
PC
P1001:35>P3.12               ; Execution will return to PROG 3, address 12
```

See Also On-line commands **B{constant}**, **LIST**, **LIST PC**, **LIST PE**, **LIST PROGRAM**, **PE**

PE

Function Report Program Execution Pointer

Scope Coordinate-system specific

Syntax **PE**

Remarks This command causes PMAC to report the motion program number and address offset of the currently *executing* programmed move in the addressed coordinate system. This is similar to the **PC** command, which reports the program number and address offset of the next move to be calculated. Since PMAC is calculating ahead in a continuous sequence of moves, **PC** will in general report a move line several moves ahead of **PE**.

If the coordinate system is not pointing to any motion program, PMAC will return an error (ERR003 if I6=1 or 3). Initially the pointing must be done with the **B{constant}** command.

Example

```
PE
P1:2
PE
P1:5
```

See Also On-line commands **B{constant}**, **LIST**, **LIST PC**, **LIST PE**, **PC**

PMATCH

Function Re-match Axis Positions to Motor Positions

Scope Coordinate-system specific

Syntax **PMATCH**

Remarks This command causes PMAC to recalculate the *axis* starting positions for the coordinate system to match the current *motor* commanded positions (by inverting the axis definition statement equations and solving for the axis position).

Normally this does not need to be done. However, if a *motor* move function, such as a jog move, an open-loop move, or a stop on abort or limit, was done since the last *axis* move or home, PMAC will not automatically know that the *axis* position has changed. If an axis move is then attempted without the use of the **PMATCH** command, PMAC will use the wrong *axis* starting point in its calculations.

Also, with an absolute sensor, a **PMATCH** command should be executed before the first programmed move, so the starting axis position matches the (non-zero) motor position.

If the **PMATCH** function is not performed, PMAC will use the last axis destination position as the starting point for its upcoming axis move calculations, which is not necessarily the same position as the current commanded motor positions.

The **PMATCH** function can be executed from within a motion program using **CMD"PMATCH"** with **DWELL**s both before and after. This is useful if the coordinate system setup changes in the middle of the program (e.g. new axis brought in, or following mode changed).

If more than one motor is defined to a given axis (as in a gantry system), the commanded position of the lower-numbered motor is used in the PMAC calculations.

> ***Note:***
>
> If I14 is set to 1, the **PMATCH** function will be executed automatically every time program execution is started. Most users will want to use I14=1 so they do not have to worry about when this needs to be done.

Example

```
#1J+.....                    ; Jog motor 1
#1J/.....                    ; Stop jogging
PMATCH.....                  ; Match axis position to current motor position
B200R.....                   ; Execute program 200

OPEN PROG 10 CLEAR
...
CMD"&1#4->100C"              ; Bring C-axis into coordinate system
DWELL100
CMD"PMATCH".                 ; Issue PMATCH so C-axis has proper start position
DWELL100
C90
```

See Also Further Position Processing (Setting Up a Motor)
Axes, Coordinate Systems (Setting Up a Coordinate System)
I-variables I14, Ix06

PR

Function Report Rotary Program Remaining

Scope Coordinate-system specific

Syntax **PR**

Remarks This command causes PMAC to report the number of program lines that have been entered in the rotary buffer for the addressed coordinate system but have not yet been executed (program remaining). This command can be useful for finding out if it is time to send new lines to the buffer. The value returned is accurate only if the rotary program buffer is open.

Example

```
B0 ..... ; Point to rotary buffer
OPEN ROT ..... ; Open rotary buffer
X10F10 ..... ; Enter 1st line
X20 ..... ; Enter 2nd line
X30 ..... ; Enter 3rd line
X40 ..... ; Enter 4th line
PR ..... ; Ask for program remaining
4 ..... ; PMAC responds that 4 lines remain
R..... ; Start running the program
PR ..... ; Ask for program remaining
2 ..... ; PMAC responds two 2 lines remain
```

See Also Rotary Program Buffers (Writing a Motion Program)
BREQ interrupt (Using Interrupts – Writing a Host Communications Program)
I-variables I16, I17

Q

Function Quit Program at End of Move

Scope Coordinate-system specific

Syntax **Q**

Remarks This causes the currently addressed coordinate system to cease execution of the program at the end of the currently executing move or the next move if that has already been calculated. The program counter is set to the next line in the program, so execution may be resumed at that point with an **R** or **S** command.

Compare this to the similar **/** command, which always stops at the end of the currently executing move..

Example

```
B10R..... ; Point to beginning of PROG 10 and run
Q..... ; Quit execution
R..... ; Resume execution
Q..... ; Quit execution again
S ; Resume execution for a single move
```

See Also Stopping Commands (Making Your Application Safe)
Control-Panel Port STEP/ Input (Connecting PMAC to the Machine)
JPAN Connector Pin 9
On-line commands **<CTRL-Q>**, **A**, **H**, **K**, **/**, ****

Q{constant}

Function Report Q-Variable Value

Scope Coordinate-system specific

Syntax **Q{constant}[..{constant}]**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the Q-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;

Remarks This command causes PMAC to report back the present value of the specified Q-variable or range of Q-variables for the addressed coordinate system.

Example
```
Q10
35
Q255
-3.4578
Q101..103
0
98.5
-0.333333333
```

See Also Q-Variables (Computational Features)
On-line commands **I{constant}**, **M{constant}**, **P{constant}**, **Q{constant}={expression}**

Q{constant}={expression}

Function Q-Variable Value Assignment

Scope Coordinate-system specific

Syntax **Q{constant}[..{constant}]={expression}**
where:

- **{constant}** is an integer from 0 to 1023 representing the number of the Q-variable;
- the optional second **{constant}** must be at least as great as the first **{constant}** – it represents the number of the end of the range;
- **{expression}** contains the value to be given to the specified Q-variable(s)

Remarks This command causes PMAC to assign the value of the expression to the specified Q-variable or range of Q-variables for the addressed coordinate system.

If a motion program buffer is open when this command is sent to PMAC it is entered into the buffer for later execution.

Example
```
Q100=2.5
Q1..10=0
```

See Also Q-Variables (Computational Features)
On-line commands **I{constant}={expression}**, **M{constant}={expression}**, **P{constant}={expression}**, **Q{constant}**
Program command **Q{constant}={expression}**

R

Function Run Motion Program

Scope Coordinate-system specific

Syntax **R**

Remarks This command causes the addressed PMAC coordinate system to start continuous execution of the motion program addressed by the coordinate system's program counter from the location of the program counter. Alternately, it will restore operation after a '****' or '**H**' command has been issued (even if a program was or is not running). Addressing of the program counter is done initially using the **B{constant** command.

The coordinate system must be in a proper condition in order for PMAC to accept this command. Otherwise, PMAC will reject this command with an error; if I6 is 1 or 3, it will report the error number. The following conditions can cause PMAC to reject this command (also listed are the remedies):

- Both limits set for a motor in coordinate system (ERR010); clear limits
- Another move in progress (ERR011); stop move (e.g. with **J**)
- Open-loop motor in coordinate system (ERR012); close loop with **J** or **A**
- Unactivated motor in coordinate system (ERR013); change Ix00remove motor from coordinate system
- No motors in the coordinate system (ERR014); put at least 1 motor in C.S.
- Fixed motion program buffer open (ERR015); close buffer and point to program
- No program pointed to (ERR015); point to program with **B** command
- Program structured improperly (ERR016); correct program structure
- Motor(s) not at same position as stopped with **/** or **** command (ERR017); move back to stopped position with **J=**

Example

```
&1B1R.....          ; C.S.1 point to PROG 1 and run
&2B200.06.....      ; C.S.2 point to N6000 of PROG 200 and run
Q.....              ; Quit this program
R.....              ; Resume running from point where stopped
H.....              ; Do a feed hold on this program
R                   ; Resume running from point where stopped
```

See Also Control Panel Port START/ Input (Connecting PMAC to the Machine)
Running a Motion Program (Writing a Motion Program)
I-variable I6
On-line commands **<CTRL-R>**, **A**, **H**, **Q**, **S**
JPAN Connector Pin 8

R[H]{address}

Function Report the contents of a specified memory address[es]

Scope Global

Syntax **R[H]{address} [,{constant}]**
where:

- **{address}** consists of a letter **X**, **Y**, or **L**; an option colon (:); and an integer value from 0 to 65535 (in hex, $0000 to $FFFF); specifying the starting PMAC memory or I/O address to be read;

- **{constant}** (optional) is an integer from 1 to 16 specifying the number of consecutive memory addresses to be read; if this is not specified, PMAC assumes a value of 1

Remarks This command causes PMAC to report the contents of the specified memory word address or range of addresses to the host (it is essentially a PEEK command). The command can specify either short (24-bit) word(s) in PMAC's X-memory, short (24-bit) word(s) in PMAC's Y-memory, or long (48-bit) words covering both X and Y memory (X-word more significant). This choice is controlled by the use of the **X**, **Y**, or **L** address prefix in the command, respectively.

If the letter **H** is used after the **R** in the command, PMAC reports back the register contents in unsigned hexadecimal form, with 6 digits for a short word, and 12 digits for a long word. If the letter **H** is not used, PMAC reports the register contents in signed decimal form.

Example

```
RHX:49152.....          ; Request contents of X-register 49152 ($C000) in hex
8F4017.....             ; PMAC responds in unsigned hex (note no '$')
RHX:$C000.....          ; Request contents of X-reg $C000 (49152) in hex
8F4017.....             ; PMAC responds in unsigned hex
RX:49152 .....          ; Request contents of same register in decimal
-7389161 .....          ; PMAC responds in signed decimal
RX:$C000 .....          ; Request contents of same register in decimal
-7389161 .....          ; PMAC responds in signed decimal
RX0 .....               ; Request contents of servo cycle counter in decimal
2953211 .....           ; PMAC responds in signed decimal
RL$0028 .....           ; Request contents of  #1 cmd. pos. reg in decimal
3072000 .....           ; PMAC responds (=1000 counts)
RHY1824,12 ...          ; Request set-up words of the conversion table
00C000 00C004 00C008 00C00C 00C010 00C014 00C018
00C01C 400723 0000295 000000 000000 ; PMAC responds in hex
```

See Also PMAC Memory Mapping (Computational Features)
On-line command **W{address}**
Memory and I/O Map Description.

RESUME PLC

Function Resume execution of specified PLC program(s).

Scope Global

Syntax

```
RESUME PLC {constant}[,{constant}...]
RES PLC {constant}[,{constant}...]
RESUME PLC {constant}[..{constant}]
RES PLC {constant}[..{constant}]
```

where:

- **{constant}** is an integer from 0 to 31, representing the program number

Remarks This command causes PMAC to resume execution of the specified uncompiled PLC program or programs at the point where execution was suspended with the **PAUSE PLC** command. This can be either at the top of the program, or at a point inside the program.

The **RESUME PLC** command cannot be used to restart execution of a PLC program that has been stopped with a **DISABLE PLC** command. However, after a PLC has been stopped with a **DISABLE PLC** command, if a **PAUSE PLC** command is then given for that PLC, then a **RESUME PLC** command can be given to start operation at the point at which it has been stopped.

Note that **RESUME PLC 0..31** will restart all PLCs that have been paused, but not any that have been disabled.

The line of the PLC at which execution will be resumed can be read with the **LIST PLC,,1** command.

PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.
If a motion or PLC program buffer is open when this command is sent to PMAC, the command will be entered into that buffer for later execution.

I-variable I5 must be in the proper state to allow the PLC program(s) specified in this command to execute.

Example

```
RESUME PLC 1
RES PLC 2,7
RESUME PLC 3,21
RESUME PLC 0..31
```

{Note: If the **RESUME** command refers to multiple PLCs, only those PLCs that have been stopped with the **PAUSE** command will be resumed.}

See Also I-variable I5
On-line commands **DISABLE PLC**, **ENABLE PLC**, **OPEN PLC**, **PAUSE PLC**, **LIST PLC**, **<CONTROL-D>**.
Program commands **DISABLE PLC**, **ENABLE PLC**, **PAUSE PLC, RESUME PLC**

S

Function Execute One Move (Step) of Motion Program

Scope Coordinate-system specific

Syntax **S**

Remarks This command causes the addressed PMAC coordinate system to start single-step execution of the motion program addressed by the coordinate system's program counter from the location of the program counter. Addressing of the program counter is done initially using the **B{constant** command.

At the default I53 value of zero, a Step command causes program execution through the next move or **DWELL** command in the program, even if this takes multiple program lines.

When I53 is set to 1, a Step command causes program execution of only a single program line, even if there is no move or **DWELL** command on that line. If there is more than one **DWELL** or **DELAY** command on a program line, a single Step command will only execute one of the **DWELL** or **DELAY** commands.

Regardless of the setting of I53, if program execution on a Step command encounters a **BLOCKSTART** statement in the program, execution will continue until a **BLOCKSTOP** statement is encountered.

If the coordinate system is already executing a motion program when this command is sent, the command puts the program in single-step mode, so execution will stop at the end of the latest calculated move. In this case, its action is the equivalent of the **Q** command.

The coordinate system must be in a proper condition in order for PMAC to accept this command. Otherwise, PMAC will reject this command with an error; if I6 is 1 or 3, it will report the error number. The same conditions that cause PMAC to reject an **R** command will cause it to reject an **S** command; refer to those conditions under the **R** command specification.

Example

```
&3B20S.....      ; C.S.3 point to beginning of PROG 20 and step
P1 .....         ; Ask for value of P1
1.....           ; PMAC responds
S.....           ; Do next step in program
P1 .....         ; Ask for value of P1 again
-3472563         ; PMAC responds –probable problem
```

See Also

Control Panel Port STEP/ Input (Connecting PMAC to the Machine)
Running a Motion Program (Writing a Motion Program)
I-variable I6, I53
On-line commands **<CTRL-S>**, **A**, **H**, **Q**, **R**, **/**, ****
Program commands **{axis}{data}**, **BLOCKSTART**, **BLOCKSTOP**, **DWELL**, **DELAY**

SAVE

Function Copy setup parameters to non-volatile memory.

Scope Global

Syntax **SAVE**

Remarks This command causes PMAC to copy setup information from active memory to non-volatile memory, so this information can be retained through power-down or reset. Its exact operation depends on the type of PMAC used.

For the "standard" PMACs with battery-backed RAM, only the basic setup information is stored with the SAVE command: I-variables, encoder conversion table entries, and VME/DPRAM address entries. This information is copied back from flash to active memory during a normal power-up/reset operation. User programs, buffers, and definitions are simply held in RAM by the battery backup; there is no need to save these.

For the option PMACs with flash-backed RAM, all user setup information, including programs, buffers, and definitions, is copied to flash memory with the **SAVE** command. This information is copied back from flash to active memory during a normal power-up/reset operation. This means that anything changed in PMAC's active memory that is not saved to flash memory will be lost in a power-on/reset cycle.

The **SAVE** operation can be inhibited by changing jumper E50 from its default state. If the **SAVE** command is issued with jumper E50 not in its default state, PMAC will report an error. The retrieval of information from non-volatile memory on power-up/reset can be inhibited by changing jumper E51 from its default state.

> ***Note:***
>
> PMAC does not provide the acknowledging handshake character to the **SAVE** command until it has finished the saving operation, a significant fraction of a second later on PMACs with battery backup and about 5 to 10 seconds on PMACs with flash backup. The host program should be prepared to wait much longer for this character than is necessary on most commands. For this reason, it is usually not a good idea to include the **SAVE** command as part of a "dump" download of a large file.

During execution of the **SAVE** command, PMAC will execute no other background tasks, including user PLCs and automatic safety checks, such as following error and overtravel limits. Particularly on boards with the flash backup where saving takes many seconds, you must make sure the system is not depending on these tasks for safety when the **SAVE** command is issued.

Example

```
I130=60000 ...          ; Set Motor 1 proportional gain
SAVE.....               ; Save to non-volatile memory
I130=80000 ...          ; Set new value
$$$.....                ; Reset card
I130.....               ; Request value of I130
60000                   ; PMAC responds with saved value
```

See Also On-line commands **$$$**, **$$$*****
Jumpers E50 and E51.

SETPHASE

Function Set motor commutation phase-position registers

Scope Global

Syntax

```
SETPHASE{constant}[,{constant}...]
SETPHASE{constant}..{constant}
[,{constant}..{constant}...]
```

where:

- **{constant}** is an integer from 1 to 8 representing a motor number

Remarks This command causes PMAC to force the commutation phase-position register for the specified motor or motor's to the value of the Ix75 phase-position offset parameter.

The main use of this command is to correct the phase position value at a known position (usually the motor home position) after an approximate phasing search or phasing read (e.g. from Hall commutation sensors). The approximate referencing is sufficient to move to a known position, but not necessarily to get peak performance from the motor.

Example

```
SETPHASE1
SETPHASE1,3,5
SETPHASE1..4
SETPHASE1..3,5..7
```

See Also Power-On Phasing Search (Setting Up PMAC Commutation)
I-variables Ix75, Ix81
Program Command **SETPHASE**

SIZE

Function Report the amount of unused buffer memory in PMAC.

Scope Global

Syntax **SIZE**

Remarks This command causes PMAC to report to the host the amount of unused long words of memory available for buffers. If no program buffer (motion, PLC or rotary buffer) is open, this value is reported as a positive number. If a buffer is currently open, the value is reported as a negative number.

Example

```
DEFINE GATHER                ; Reserve all remaining memory for gathering
SIZE.....                    ; Ask for amount of open memory
0.....                       ; PMAC reports none available
DELETE GATHER                ; Free up memory from gathering buffer
SIZE.....                    ; Ask for amount of open memory
41301 .....                  ; PMAC reports number of words available
OPEN PROG 10                 ; Open a motion program buffer
SIZE.....                    ; Ask for amount of open memory
-41302                       ; The negative sign shows a buffer is open
```

See Also I-Variable I18
On-line commands **DELETE GATHER**, **DELETE TRACE**

TYPE

Function Report type of PMAC

Scope Global

Syntax **TYPE**

Remarks This command causes PMAC to return a string reporting the configuration of the card. It will report the configuration as a text string in the format:

{PMAC type},{Bus type},{Backup type},{Servo Type},{Ladder type},{Clock Multiplier}
where
{PMAC type}:

PMAC1	First generation PMAC (including PMAC"1.5")
PMAC2	Second generation PMAC
PMACUL	Ultra-lite (MACRO only PMAC2)

{Bus type}:

ISA	IBM-PC ISA bus
VME	VME bus
STD	STD bus
ISA/VME	PMAC1 firmware can support both busses

{Backup type}:

BATTERY	Battery-backed RAM
FLASH	AMD-style flash-backed RAM
I-FLASH	Intel-style flash-backed RAM

{Servo type}:

PID	Standard PID servo algorithm
ESA	Option 6 Extended servo algorithm

{Ladder type}

{blank} no ladder-logic diagram support
LDs Ladder-logic diagram support

{Clock multiplier}:
CLK Xn where n is the multiplication of crystal frequency to CPU frequency

Example
```
TYPE
PMAC1, ISA/VME, BATTERY, PID, CLK X1
TYPE
PMAC2, ISA, FLASH, ESA, CLK X3
TYPE
PMACUL, VME, FLASH, PID, LDs, CLK X2
```

See Also On-line commands **VERSION, DATE**

UNDEFINE

Function Erase Coordinate System Definition

Scope Coordinate-system specific

Syntax
```
UNDEFINE
UNDEF
```

Remarks This command causes PMAC to erase all of the axis definition statements in the addressed coordinate system. It does not affect the axis definition statements in any other coordinate systems. It can be useful to use before making new axis definitions.

To erase the axis definition statement of a single motor only, use the **#{constant}->0** command; to erase all the axis definition statements in every coordinate system, use the **UNDEFINE ALL** command.

Example
```
&1 ..... ; Address C.S.1
#1->..... ; Ask for axis definition of Motor 1
10000X..... ; PMAC responds
#2->..... ; Ask for axis definition of Motor 2
10000Y..... ; PMAC responds
UNDEFINE..... ; Erase axis definitions
&2 ..... ; Address C.S.2
#1->10000X... ; Redefine Motor 1 as X-axis in C.S.2
#2->10000Y ; Redefine Motor 2 as Y-axis in C.S.2
```

See Also Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **#{constant}->**, **#{constant}->0**, **UNDEFINE ALL**

UNDEFINE ALL

Function Erase coordinate definitions in all coordinate systems

Scope Global

Syntax
```
UNDEFINE ALL
UNDEF ALL
```

Remarks This command causes all of the axis definition statements in all coordinate systems to be cleared. It is a useful way of starting over on a reload of PMAC's coordinate system definitions.

Example
```
&1#1->..... ; Request axis definition of Motor 1 in C.S. 1
1000X..... ; PMAC responds
&2#5->..... ; Request axis definition of Motor 5 in C.S. 2
```

```
1000X.....                    ; PMAC responds
UNDEFINE ALL                  ; Erase all axis definitions
&1#1->.....                   ; Request axis definition of Motor 1 in C.S. 1
0.....                        ; PMAC responds that there is no definition
&2#5->.....                   ; Request axis definition of Motor 5 in C.S. 2
0                             ; PMAC responds that there is no definition
```

See Also Axes, Coordinate Systems (Setting Up a Coordinate System)
On-line commands **#{constant}->0**, **#{constant}->**, **UNDEFINE**.

V

Function Report motor velocity

Scope Motor specific

Syntax **V**

Remarks This command causes PMAC to report the present actual motor velocity to the host, scaled in counts/servo cycle, rounded to the nearest tenth. It is reporting the contents of the motor actual velocity register (divided by [Ix09*32]).

To convert this reported value to counts/msec, multiply by 8,388,608*(Ix60+1) and divide by I10. It can be further converted to engineering units with additional scaling constants.

> ***Note:***
>
> The velocity values reported here are obtained by subtracting positions of consecutive servo cycles. As such, they can be very noisy. For purposes of display, it is probably better to use averaged velocity values held in registers Y:$082A, Y:$08EA, etc., accessed with M-variables

Example

```
V.....                        ; Request actual velocity of addressed motor
21.9.....                     ; PMAC responds with 21.9 cts/cycle
.....                         ; (*8,388,608/3,713,707 = 49.5 cts/msec)
#6V.....                      ; Request velocity of Motor 6
-4.2.....                     ; PMAC responds
#5V#2V.....                   ; Request velocities of Motors 5 and 2
0.....                        ; PMAC responds with Motor 5 first
7.6                           ; PMAC responds with Motor 2 second
```

See Also I-variables I10, Ix09, Ix60
On-line commands **<CTRL-V>**, **F**, **P**
Memory map registers X:$0033, X:$006F, etc., X:$082A, X:$08EA, etc.
Suggested M-variable definitions Mx66

VERSION

Function Report PROM firmware version number

Scope Global

Syntax **VERSION**
VER

Remarks This command causes PMAC to report the firmware version it is using.

When a flash-memory PMAC is in "bootstrap mode" (powering up with E51 ON), PMAC will report the version of the bootstrap firmware, not the operational firmware. Otherwise, it will report the operational firmware version. To change from bootstrap mode to normal operational mode, use the **<CTRL-R>** command.

Example
```
VERSION .....                ; Ask PMAC for firmware version
1.12D                        ; PMAC responds
```

See Also Resetting PMAC (Talking to PMAC)
On-line command **DATE**, **TYPE**

W{address}

Function Write value(s) to a specified address(es).

Scope Global

Syntax **W{address,{constant} [,{constant}...]**
where:

- **{address}** consists of a letter **X**, **Y**, or **L**; an option colon (:); and an integer value from 0 to 65535 (in hex, $0000 to $FFFF); specifying the starting PMAC memory or I/O address to be read;
- **{constant}** is an integer, specified in decimal or hexadecimal, specifying the value to be written to the specified address;
- further **{constants}** specify integer values to be written into subsequent consecutive higher addresses

Remarks This command causes PMAC to write the specified **{constant}** value to the specified memory word address, or if a series of **{constant}** values is specified, to write them to consecutive memory locations starting at the specified address (it is essentially a memory POKE command). The command can specify either short (24-bit) word(s) in PMAC's X-memory, short (24-bit) word(s) in PMAC's Y-memory, or long (48-bit) words covering both X and Y memory (X-word more significant). This choice is controlled by the use of the **X**, **Y**, or **L** address prefix in the command, respectively.

Example
```
WY:$C002,4194304             ; This should put 5V on DAC2 (provided I200=0 so servo
                             ; does not overwrite)
WY$720,$00C000,$00C004,$00C008,$00C00C
                             ; This writes the first four entries to the encoder
                             ; conversion table
```

See Also On-line command **R[H]{address}**
Memory and I/O Map.

Z

Function Make commanded axis positions zero.

Scope Coordinate-system specific

Syntax **Z**

Remarks This command causes PMAC to re-label the current commanded axis positions for all axes in the coordinate system as zero. It does not cause any movement; it merely re-names the current position.

This command is simply a short way of executing **{axis}=0** for all axes in the coordinate system. **PSET X0 Y0** (etc.) is the equivalent motion program command.

This does not set the *motor* position registers to zero; it changes motor position bias registers to reflect the new offset between motor zero positions and axis zero positions. However, the motor reported positions will reflect the new bias, and report positions of zero (+/- the following error)

Example

```
<CTRL-P> ...                          ; Ask for reported motor positions
2001 5002 3000 0 0 0 0 0              ; PMAC reports positions
Z.....                                ; Zero axis positions
<CTRL-P>.....                         ; Ask for reported motor positions again
1 2 -1 0 0 0 0 0                      ; PMAC responds
```

See Also

On-line commands **{axis}={constant}**
Memory map registers D:$0813, D:$08D3, etc.
Suggested M-variable definitions Mx64.

PMAC PROGRAM COMMAND SPECIFICATION

{axis}{data}[{axis}{data}...]

Function Position-Only Move Specification

Type Motion program (PROG and ROT)

Syntax **`{axis}{data}[{axis}{data}...]`**
where:

- **`{axis}`** is the character specifying which axis (X, Y, Z, A, B, C, U, V, W).
- **`{data}`** is a constant (no parentheses) or an expression (in parentheses) representing the end position or distance.
- **`[{axis}{data}...]`** is the optional specification of simultaneous movement for more axes.

Remarks This is the basic PMAC move specification statement. It consists of one or more groupings of an axis label and its associated value. The value for an axis is scaled (units determined by the axis definition statement). It represents a position if the axis is in absolute (ABS) mode, or a distance if the axis is in incremental (INC) mode. The order in which the axes are specified does not matter.

This command tells the axes *where* to move; it does not tell them *how* to move there. (Other program commands and parameters define how. These must be set up ahead of time.)

The type of motion a given command causes is dependent on the mode of motion and the state of the system at the beginning of the move.

Example

```
X1000
X(P1+P2)
Y(Q100+500) Z35 C(P100)
X1000 Y1000
A(P1) B(P2) C(P3)
X(Q1*SIN(Q2/Q3)) U500
```

See Also Axis Definition Statements (Setting Up a Coordinate System)
Motion program commands **`LINEAR`**, **`CIRCLEn`**, **`RAPID`**, **`SPLINE1`**, **`PVT`**
Motion program commands **`TA`**, **`TS`**, **`TM`**, **`F`**, **`ABS`**, **`INC`**
I-variables Ix87, Ix88, Ix89, Ix90

{axis}{data}:{data} [{axis}{data}:{data}...]

Function Position and Velocity Move Specification

Type Motion program (PROG and ROT)

Syntax **`{axis}{data}:{data} [{axis}{data}:{data}...]`**
where:

- **`{axis}`** is the character specifying which axis (X, Y, Z, A, B, C, U, V, W);
- **`{data}`** is a constant (no parentheses) or an expression (in parentheses) representing the end position or distance;
  - **`:{data}`** represents the ending velocity
  - **`[{axis}{data}:{data}...]`** is the optional specification of simultaneous movement for more axes.

Remarks In the case of PVT (position, velocity, time) motion mode, both the ending position and velocity are specified for each segment of each axis. The command consists of one or more groupings of axes labels with two data items separated by a colon character.

The first data item for each axis is the scaled ending position or distance (depending on whether the axis is in absolute [**ABS** or incremental [**INC**] mode. Position scaling is determined by the axis definition statement). The second data item (after the colon) is the ending velocity.

The velocity units are the scaled position units as established by the axis definition statements divided by the time units as set by Ix90 for Coordinate System x. The velocity here is a signed quantity, not just a magnitude. See the examples in the PVT mode description of the Writing a Motion Program section.

The time for the segment is the argument for the most recently executed **PVT** or **TA** command, rounded to the nearest millisecond.

In PVT mode, if no velocity is given for the segment, PMAC assumes an ending velocity of zero for the segment.

Example

```
X1000:50
Y500:-32 Z737.2:68.93
A(P1+P2):(P3) B(SIN(Q1)):0
```

See Also PVT Mode Moves (Writing a Motion Program)
Axis Definition Statements (Setting Up a Coordinate System)
I-variables Ix87, Ix90
Motion program commands **PVT**, **TA**

{axis}{data}^{data}[{axis}{data}^{data}...]

Function Move Until Trigger

Type Motion program

Syntax **{axis}{data}^{data}[{axis}{data}^{data}...]**
where:

- **{axis}** is the character specifying which axis (X, Y, Z, A, B, C, U, V, W).
- the first **{data}** is a constant (no parentheses) or expression (in parentheses) representing the end position or distance in the absence of a trigger.
- the second **{data}** (after the ^ arrow) is a constant (no parentheses) or expression (in parentheses) representing the distance from the trigger position.
- **[{axis}{data}^{data}...]** is the optional specification of simultaneous movement for more axes.

Remarks In the **RAPID** move mode, this move specification permits a move-until-trigger function. The first part of the move description for an axis (before the ^ sign) specifies where to move in the absence of a trigger. It is a position if the axis is in absolute mode; it is a distance if the axis is in incremental mode. In both cases the units are the scaled axis user units. If no trigger is found before this destination is reached, the move is a standard **RAPID** move.

The second part of the move description for an axis (after the ^ sign) specifies the distance from the trigger position to end the post-trigger move if a trigger is found. The distance is expressed in the scaled axis user units.

Each motor assigned to an axis specified in the command, executes a separate move-until-trigger. All the assigned motors will start together, but each can have its own trigger condition. If a common trigger is required, the trigger signal must be wired into all motor interfaces. Each motor can finish at a separate time. The next line in the program will not start to execute until all motors have finished their moves. No blending into the next move is possible.

The trigger for a motor can be either a hardware input trigger (if bit 17 of Ix03 is 0), or the motor warning following error status bit (if bit 17 of Ix03) is 1. Bit 16 of Ix03 should also be set to 1 in this case. If a hardware input trigger is used, Encoder/Flag I-variables 2 and 3 (e.g. I902 and I903) for the flag channel specified by Ix25 determine which edge(s) of which flag(s) cause the trigger. If the warning following error bit is used for torque-limited triggering, then Ix12 sets the size of the warning following error.

The speed of the move, both before the trigger and after, is set by Ix22 if I50=0 or by Ix16 if I50=1. The acceleration is set by Ix19 to Ix21.

On the same line, some axes may be specified for normal untriggered **RAPID** moves that will execute simultaneously.

If the move ends for a motor without a trigger being found, the "trigger move" status bit (bit 7 of the second motor status word returned on a **?** command) is left set after the end of the move. If the trigger has been found, this bit is cleared to 0 at the end of the move.

Example

```
X1000^0
X10^-0.01 Y5.43^0.05
A(P1)^(P2) B10^200 C(P3)^0 X10
```

See Also

Move-Until-Trigger (Writing a Motion Program)
Torque-Limited Triggering (Setting Up a Motor)
RAPID-mode moves (Writing a Motion Program)

{axis}{data} [{axis}{data}...] {vector}{data} [{vector}{data}...]

Function Circular Arc Move Specification

Type Motion program (PROG and ROT)

Syntax **{axis}{data} [{axis}{data}...] {vector}{data} [{vector}{data}...]**

where:

- **{axis}** is a character specifying which axis (X, Y, Z, A, B, C, U, V, W).
- **{data}** is a constant (no parentheses) or an expression (in parentheses) representing the end position or distance.
- **[{axis}{data}...]** is the optional specification of simultaneous movement for more axes.
- **{vector}** is a character (I, J, or K) specifying a vector component (parallel to the X, Y, or Z axis, respectively) to the center of the arc; or the character R specifying the magnitude of the vector.
- **{data}** specifies the magnitude of the vector component.
- **[{vector}{data}...]** is the optional specification of more vector components.

Remarks

For a blended circular mode move, both the move endpoint and the vector to the arc center are specified. The endpoint is specified just as in a **LINEAR** mode move, either by position (referenced to the coordinate system origin), or distance (referenced to the starting position).

The center of the arc for a circular move must also be specified in the move command. This is usually done by defining the vector to the center. This vector can either be referenced to the starting point of the move (incremental radial vector mode – the default, or if an **INC (R)** command has been given), or it can be referenced to the coordinate system origin (absolute radial vector mode – if an **ABS (R)** command has been given).

Alternatively, just the magnitude of the vector to the center can be specified with **R{data}** on the command line. If this is the case, PMAC will calculate the location of the center itself. If the value specified by **{data}** is positive, PMAC will compute the short arc path to the destination (<= 180^{o}); if it is negative, PMAC will compute the long arc path (>= 180^{o}). It is not possible to specify a full circle in one command with the R vector specifier.

The plane for the circular arc must have been defined by the **NORMAL** command (the default – **NORMAL K-1** – defines the XY plane). This command can only define planes in XYZ-space, which means that only the X, Y, and Z axes can be used for circular interpolation. Other axes specified in the same move command will be interpolated linearly to finish in the same time.

The direction of the arc to the destination point – clockwise or counterclockwise – is controlled by whether the card is in **CIRCLE1** (clockwise) or **CIRCLE2** (counterclockwise) mode. The sense of clockwise in the plane is determined by the direction of the **NORMAL** vector to the plane.

If the destination point is a different distance from the center point than is the starting point, the radius is changed smoothly through the course of the move, creating a spiral. This is useful in compensating for any roundoff errors in the specifications. However, if the distance from either the starting point or the destination point to the center point is zero, an error condition will be generated and the program will stop.

If the vector from the starting point to the center point does not lie in the circular interpolation plane, the projection of that vector into the plane is used. If the destination point does not lie in the same circular interpolation plane as the starting point, a helical move is done to the destination point.

If the destination point (or its projection into the circular interpolation plane containing the starting point) is the same as the starting point, a full 360^{o} arc is made in the specified direction (provided that IJK vector specification is used). In this case, only the vector needs to be specified in the move command, because for any axis whose destination is not specified, the destination point is automatically taken to be the same as the starting point.

If no vector, and no radial magnitude is specified in the move command, a linear move will be done to the destination point, even if the program is in circular mode.

Note:

PMAC performs arc moves by segmenting the arc and performing the best cubic fit on each segment. I-variable I13 determines the time for each segment. I13 must be set greater than zero to put PMAC into this segmentation mode in order for arc moves to be done. If I13 is set to zero, circular arc moves will be done in linear fashion.

Example

```
X5000 Y3000 I1000 J1000
X(P101) Z(P102) I(P201) K(P202)
X10 I5
X10 Y20 C5 I5 J5
Y5 Z3 R2
J10                                    ; Specifies a full circle of 10 unit radius
```

See Also Circular Moves (Writing a Motion Program)
I-variables I13, Ix87, Ix88, Ix89, Ix90
Program commands **NORMAL**, **ABS**, **INC**, **CIRCLE1**, **CIRCLE2**, **TA**, **TS**, **TM**, **F**

A{data}

Function A-Axis Move

Type Motion program (PROG or ROT)

Syntax **A{data}**
where:

- **{data}** is a floating-point constant or expression representing the position or distance in user units for the U-axis.

Remarks This command causes a move of the A-axis. See **{axis}{data}** descriptions above.

Example

```
A10
A(P23)
A25 B10 Z35
A(20*SIN(Q5))
```

See Also Program commands **{axis}{data}**, **B**, **C**, **U**, **V**, **W**, **X**, **Y**, **Z**, **CALL**, **READ**

ABS

Function Absolute Move Mode

Type Motion program (PROG and ROT)

Syntax **ABS [({axis}[,{axis}...])]**
where:

- **{axis}** is a character (X,Y,Z,A,B,C,U,V,W) representing the axis to be specified, or the character R to specify radial vector mode.

> ***Note:***
>
> No spaces are permitted in this command.

Remarks The **ABS** command without arguments causes all subsequent positions in motion commands for all axes in the coordinate system running the motion program to be treated as absolute positions. This is known as absolute mode, and it is the power-on default condition.
An **ABS** statement with arguments causes the specified axes in the coordinate system running the program to be in absolute mode, and all others remain the same.
If **R** is specified as one of the 'axes', the I, J, and K terms of the circular move radius vector specification will be specified in absolute form (i.e., as a vector from the origin, not from the move start point). An **ABS** command without any arguments does not affect this vector specification. The default radial vector specification is incremental.
If no motion program buffer is open when this command is sent to PMAC, it will be executed as an on-line coordinate system command.

Example

```
ABS(X,Y)
ABS
ABS(V)
ABS(R)
```

See Also Circular Moves (Writing a Motion Program)
On-line commands **ABS**, **INC**.
Program commands **{axis}{data}**, **{axis}{data}{vector}{data}**, **INC**.

ADDRESS

Function Motor/Coordinate System Modal Addressing

Type PLC programs 1 to 31 **only**

Syntax
```
ADDRESS [#{constant}][&{constant}]
ADR [#{constant}][&{constant}]
```
where:

- **{constant}** is an integer constant from 1 to 8 representing the motor (#) number or the coordinate system (&) number to be addressed.

Remarks This statement, when executed, sets the motor and/or coordinate system that will be addressed by this particular PLC program when it commands motor- or coordinate-system-specific commands with no addressing in those commands. The addressed coordinate system also controls which set of Q-variables is accessed, even for ATAN2 functions which automatically use Q0.

This command does not affect host addressing, the addressing of other PLC programs, or the selection of the control panel inputs. The addressing stays in effect until another **ADDRESS** statement supersedes it. Default addressing at power-on/reset is #1 and &1.

In motion programs, there is no modal addressing for **COMMAND** statements; each **COMMAND** statement must contain the motor or coordinate-system specifier within its quotation marks. A motion program automatically operates on the Q-variables of the coordinate system executing the program.

Example
```
ADDRESS &4
ADR #2
ADDRESS &2#2
ADR#1                    ; Modally address Motor 1
CMD"J+"                  ; This will start Motor 1 jogging
CMD"#2J+"                ; This will start Motor 2 jogging
CMD"J/"                  ; This will stop Motor 1
```

See Also Addressing Modes (Talking To PMAC)
Q-Variables (Program Computational Features)
Program commands **COMMAND**, **Q{constant}={expression}**

ADIS{constant}

Function Absolute displacement of X, Y, and Z axes

Type Motion program (PROG and ROT)

Syntax
```
ADIS{constant}
```
where:

- **{constant}** is an integer constant representing the number of the first of three consecutive Q-variables to be used in the displacement vector.

Remarks This command loads the currently selected (with **TSEL**) transformation matrix for the coordinate system with offset values contained in the three Q-variables starting with the specified one. This has the effect of renaming the current commanded X, Y, and Z axis positions (from the latest programmed move) to the values of these variables (X=Q{data}, Y=Q({data}+1), Z=Q({data}+2)).This command does not cause any movement of any axes; it simply renames the present positions.

This command is almost equivalent to a **PSET X(Q{data}) Y(Q({data}+1)) Z(Q({data}+2))** command, except that **ADIS** does not force a stop between moves as **PSET** does.

Example

```
Q20=7.5
Q21=12.5
Q22=25
ADIS 20                    ; This makes the current X position 7.5, Y 12.5, Z25
```

See Also

Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **AROT**, **IDIS**, **IROT**, **TINIT**

AND ({condition})

Function Conditional AND

Type PLC program only

Syntax **AND ({condition})**
where:

- **{condition}** is a simple or compound condition

Remarks This statement forms part of an extended compound **IF** or **WHILE** condition to be evaluated in a PLC program. It must immediately follow an **IF**, **WHILE**, **AND**, or **OR** statement This **AND** is a Boolean operator logically combining the full conditions on its line and the program line immediately above. It takes lower precedence than **AND** or **OR** operators within a compound condition on a single line (the parentheses cause those to be executed first), but it takes higher precedence than an **OR** operator that starts a line.

In motion programs, there can be compound conditions within one program line, but not across multiple program lines, so this statement is not permitted in motion programs.

> ***Note:***
>
> This logical **AND** command, which acts on condition should not be confused with the bit-by-bit **&** (ampersand) operator that acts on values.

Example

```
IF (M11=1)                 ; This branch will start a motion program running
AND (M12=1)                ; on a cycle where inputs M11 and M12 are 1 and
AND (M21=0)                ; M21 is still zero. Note that M21 is immediately
CMD"R"                     ; set to one so the run command will not be given
M21=1                      ; again in the next cycle.
ENDIF
```

See Also

Writing a PLC Program
Conditions (Program Computational Features)
Program Commands **IF**, **WHILE**, **OR**

AROT{constant}

Function Absolute rotation/scaling of X, Y, and Z axes

Type Motion program (PROG and ROT)

Syntax **AROT{constant}**
where:

- **{constant}** is an integer representing the number of the first of nine consecutive Q-variables to be used in the rotation/scaling matrix.

Remarks This command loads the currently selected (with **TSEL**) transformation matrix for the coordinate system with rotation/scaling values contained in the nine Q-variables starting with the specified one. This has the effect of renaming the current commanded X, Y, and Z axis positions (from the latest programmed move) by multiplying the XYZ vector by this matrix.

The rotation and scaling is done relative to the "base" XYZ coordinate system, defined by the axis definition statements. The math performed is:

$$[Xrot\ Yrot\ Zrot]^T = [Rot\ Matrix]\ [Xbase\ Ybase\ Zbase]^T$$

This command does not cause any movement of any axes. It simply renames the present positions.

> ***Note:***
>
> When using this command to scale the coordinate system, use the **I, J, K** vector specification for circle commands. Do not use the radius center specification. The radius does not get scaled.

Example Create a 3x3 matrix to rotate the XY plane by 30 degrees about the origin

```
Q40=COS(30) Q41=SIN(30) Q42=0
Q43=-SIN(30)       Q44=COS(30) Q45=0
Q46=0              Q47=0              Q48=1
AROT 40                                          ; Implement the change
```

Create a 3x3 matrix to scale the XYZ space by a factor of 3

```
Q50=3       Q51=0              Q52=0
Q53=0       Q54=3              Q55=0
Q56=0       Q57=0              Q58=3
AROT 50                                          ; Implement the change
```

See Also Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **IDIS**, **IROT**, **TINIT**

B{data}

Function B-Axis Move

Type Motion program (PROG and ROT)

Syntax **B{data}**
where:

- **{data}** is a floating-point constant or expression representing the position or distance in user units for the U-axis.

Remarks This command causes a move of the B-axis. (See **{axis}{data}** description, above.)

See Also Program commands **{axis}{data}, A, C, U, V, W, X, Y, Z, CALL, READ**

BLOCKSTART

Function Mark Start of Stepping Block

Type Motion program (PROG and ROT)

Syntax **BLOCKSTART**
BSTART

Remarks This statement allows for multiple moves to be done on a single step command. Execution on a step command will proceed until the next **BLOCKSTOP** statement in the program (without **BLOCKSTART**, only a single servo command is executed on a step command). Also, if Ix92=1 (move blending disabled), all moves between **BLOCKSTART** and **BLOCKSTOP** will be blended together. This does not affect how a program is executed from a run command if Ix92=0.

This structure is particularly useful for executing a single sequence of PVT mode moves, because the individual segments do not end at zero velocity, making normal stepping very difficult.

Example For the program segment:

```
BLOCKSTART
INC
X10:100
X20:100
X20:100
X10:0
BLOCKSTOP
```

All four move segments will be executed on a single **S** command.

See Also I-variable Ix92
On-line commands **<CONTROL-S>**, **R**, **S**.
Program commands **BLOCKSTOP**, **STOP**

BLOCKSTOP

Function Mark End of Stepping Block

Type Motion program (PROG and ROT)

Syntax

```
BLOCKSTOP
BSTOP
```

Remarks This statement marks the end of the block of statements, begun with a **BLOCKSTART**, to be done on a single 'step' command, or to be blended together even if Ix92=1 (move blending disabled). This does not affect how a program is executed from a 'run' command if Ix92=1.

Example See example under **BLOCKSTART** above.

See Also I-variable Ix92
On-line commands **<CONTROL-S>**, **R**, **S**
Program commands **BLOCKSTART**, **STOP**

C{data}

Function C-Axis Move

Type Motion program (PROG and ROT)

Syntax **C{data}**
where:

- **{data}** is a floating-point constant or expression representing the position or distance in user units for the U-axis.

Remarks This command causes a move of the C-axis. (See **{axis}{data}** description above.)

See Also Program commands **{axis}{data}**, **A**, **B**, **U**, **V**, **W**, **X**, **Y**, **Z**, **CALL**, **READ**

CALL

Function Jump to Subprogram With Return

Type Motion program (PROG and ROT)

Syntax **CALL{data} [{letter}{data}...]**
where:

- the first **{data}** is a floating-point constant or expression from 1.00000 to 32767.99999, with the integer part representing the motion program number to be called, and the fractional part representing the line label (**N** or **O**) within the program to be called (the line label number is equal to the fractional part multiplied by 100,000; every motion program has an implicit **N0** at the top).
- **{letter}** is any letter of the English alphabet, except N or O, representing the variable into which the value following it will be placed (Q101 to Q126 for A to Z respectively).
- following **{data}** is a floating-point constant or expression representing the value to be put into the variable.

Remarks This command allows the program to execute a subprogram and then return execution to the next line in the program. A subprogram is entered into PMAC the same as a program, and is labeled as PROGn (so one program can call another as a subprogram). The number **n** of the PROG heading is the one to which the value after **CALL** refers: **CALL7** would execute **PROG7** and return.

The value immediately following **CALL** can take fractional values. If there is no fractional component, the called program starts at the beginning. If there is a fractional component, the called program is entered at a line label specified by the fractional component (if this label does not exist, PMAC will generate an error and stop execution). PMAC works with five fractional digits to specify the line label; if fewer are used, it automatically fills out the rest with zeros. For instance, **CALL 35.1** is interpreted as **CALL 35.10000**, which causes a jump to label **N10000** of program 35. **CALL 47.123** causes a jump to label **N12300** of program 47.

If letters and data (e.g. **X1000**) follow the **CALL{data}**, these can be arguments to be passed to the subprogram. If arguments are to be passed, the first line executed in the subroutine should be a **READ** statement. This statement will take the values associated with the specified letters and place them in the appropriate Q-variable. For instance, the data following **A** is placed in variable Q101 for the coordinate system executing the program; that following **B** is placed in Q102; and so on, until the data following **Z** is placed in Q126. The subprogram can then use these variables.

If the subprogram calls another subprogram with arguments, the same Q-variables are used. Refer to **READ** for more details.

If there is no **READ** statement in the subroutine, or if not all the letter values in the **CALL** line are read (the **READ** statement stops as soon as it sees a letter in the calling line that is not in its list of letters to read), the remaining letter commands are executed upon return from the subroutine. For example, **G01 X10 Y10** is equivalent to a **CALL 1000.01 X10 Y10**. To implement the normal function for **G01** (linear move mode), there would be the following subroutine in PROG 1000:

```
N1000 LINEAR RETURN
```

Upon the return, **X10 Y10** would be executed as a move according to the move mode in force, which is **LINEAR**.

If the specified program and/or line label do not exist, the **CALL** command is ignored, and the program continues as if it were not there. No error is generated.

Example

```
CALL500                 ; to Prog 500 at the top (N0)
CALL500.1               ; to Prog 500 label N10000
CALL500.12              ; to Prog 500 label N12000
CALL500.123             ; to Prog 500 label N12300
CALL500.1234            ; to Prog 500 label N12340
CALL500.12345           ; to Prog 500 label N12345
CALL700 D10 E20         ; to Prog 700 passing D and E
```

See Also

On-line command **B{constant}**
Program commands **GOTO**, **GOSUB**, **READ**, **RETURN**, **G{data}**, **M{data}**, **T{data}**, **D{data}**, **N{constant}**, **O{constant}**, **PRELUDE**

CC0

Function Turn Off Cutter Radius Compensation

Type Motion program (PROG and ROT)

Syntax **CC0**

Remarks This turns off the cutter radius compensation mode, reducing it gradually through the next move. This is equivalent to the **G40** command of the machine-tool standard RS-274 language.

Example

```
CCR0.5                  ; 1/2 unit cutter radius
CC1                     ; Cutter compensation on to the left
X10 Y10                 ; Compensation introduced during this move
X10 Y20
X20 Y20
X20 Y10
X10 Y10
CC0                     ; Cutter compensation off
X0 Y0                   ; Compensation eliminated during this move
OPEN PROG 1000          ; G-Code Subprogram
  ...
N40000 CC0 RETURN       ; To implement G40 directly in PMAC
```

See Also

Cutter (Tool) Radius Compensation
Program commands **CC1**, **CC2**, **CCR{data}**

CC1

Function Turn On Cutter Radius Compensation Left

Type Motion program (PROG and ROT)

Syntax **CC1**

Remarks This turns on the cutter radius compensation mode, introducing the compensation gradually through the next move. The cutter is offset to the left of the programmed tool path, looking in the direction of cutter movement. The plane of the compensation is determined by the **NORMAL** command. This is equivalent to the **G41**Error! Bookmark not defined. command of the machine-tool standard RS-274

language.

Example

```
CCR0.25                          ; 1/4 unit cutter radius
CC1                              ; Cutter compensation on to the left
X10 Y10                          ; Compensation introduced during this move
X10 Y20
X20 Y20
X20 Y10
X10 Y10
CC0                              ; Cutter compensation off
X0 Y0                            ; Compensation eliminated during this move
OPEN PROG 1000                   ; G-Code Subprogram
  ...
N41000 CC1 RETURN                ; To implement G41 directly in PMAC
```

See Also Cutter (Tool) Radius Compensation
Program commands **CC2**, **CC0**, **CCR{data}**, **NORMAL**

CC2

Function Turn On Cutter Radius Compensation Right

Type Motion program (PROG and ROT)

Syntax **CC2**

Remarks This turns on the cutter radius compensation mode, introducing the compensation gradually through the next move. The cutter is offset to the right of the programmed tool path, looking in the direction of cutter movement. The plane of the compensation is determined by the **NORMAL** command. This is equivalent to the **G42** command of the machine-tool standard RS-274 language.

Example

```
CCR1.5                           ; 1-1/2 unit cutter radius
CC2                              ; Cutter compensation on to the right
X10 Y10                          ; Compensation introduced during this move
X10 Y20
X20 Y20
X20 Y10
X10 Y10
CC0                              ; Cutter compensation off
X0 Y0                            ; Compensation eliminated during this move
OPEN PROG 1000                   ; G-Code Subprogram
  ...
N42000 CC2 RETURN                ; To implement G42 directly in PMAC
```

See Also Tool Radius Compensation
Program commands **CC1**, **CC0**, **CCR**, **NORMAL**

CCR{data}

Function Set Cutter Compensation Radius

Type Motion program (PROG and ROT)

Syntax **CCR{data}**
where:

- **{data}** represents the cutter compensation radius to be used, in user length units.

Remarks This sets the radius for cutter compensation (when on) in user units, as defined by the axis definition statements for the X, Y, and Z axes for the coordinate system. This function is often part of the **D** tool data used in the machine-tool standard RS-274 (G)

code.

See Also Program commands **CC1, CC2, CC0, D{data}, NORMAL**

CIRCLE1

Function Set Blended Clockwise Circular Move Mode

Type Motion program (PROG and ROT)

Syntax
CIRCLE1
CIR1

Remarks This command puts the program into clockwise circular move mode. The plane for the circular interpolation is defined by the most recent **NORMAL** command, which has also defined the sense of clockwise and counterclockwise in the plane.

The program is taken out of this circular move mode by another move mode command: the other **CIRCLE** mode, **LINEAR**, **PVT**, **RAPID**, etc. Any circular move command must have either an **R** or an **IJK** vector specification. Otherwise it will be performed as a linear move even when in **CIRCLE** mode.

Note:

PMAC must be in move segmentation mode (I13>0) in order to perform circular interpolation. If I13=0 (no move segmentation), the moves will be linearly interpolated.

Example

```
LINEAR                  ; Linear interpolation mode
X10Y10 F2               ; Linear move
CIRCLE1                 ; Clockwise circular interpolation mode
X20 Y20 I10             ; Arc of 10-unit radius
X25 Y15 J-5             ; Arc of 5-unit radius
LINEAR                  ; Go back to linear mode
X25 Y5                  ; Linear move
```

See Also Circular Moves (Writing a Motion Program)
I-variable I13
Program commands **NORMAL**, **CIRCLE2**, **LINEAR**, **PVT**, **RAPID**, **SPLINE1**, **{axis}{data}{vector}{data}**

CIRCLE2

Function Set Blended Counterclockwise Circular Move Mode

Type Motion program (PROG and ROT)

Syntax
CIRCLE2
CIR2

Remarks The **CIRCLE2** command puts the program into counterclockwise circular move mode. The plane for the circular interpolation is defined by the most recent **NORMAL** command, which has also defined the sense of clockwise and counterclockwise in the plane.

The program is taken out of this circular move mode by another move mode command: the other **CIRCLE** mode, **LINEAR**, **PVT**, **RAPID**, etc. Any circular move command must have either an **R** or an **IJK** vector specification. Otherwise it will be performed as a linear move even when in **CIRCLE** mode.

Note:

PMAC must be in move segmentation mode (I13>0) in order to perform circular interpolation. If I13=0 (no move segmentation), the moves will

be linearly interpolated.

Example

```
LINEAR                  ; Linear interpolation mode
X10Y0 F2                ; Linear move
CIRCLE2                 ; Counterclockwise circular interpolation mode
X20 Y10 J10             ; Arc of 10-unit radius
X15 Y15 I-5             ; Arc of 5-unit radius
CIRCLE1                 ; Clockwise circle mode
X5 Y25 J10              ; Arc move of 10-unit radius
```

See Also Circular Moves (Writing a Motion Program)
I-variable I13
Program commands **NORMAL**, **CIRCLE1**, **LINEAR**, **PVT**, **RAPID**, **SPLINE1**, **{axis}{data}{vector}{data}**

COMMAND "{command}"

Function Program Command Issuance

Type Motion program (PROG and ROT); PLC program

Syntax

```
COMMAND "{command}"
CMD "{command}"
```

Remarks This statement causes the program to issue a command to PMAC as if it came from the host (except for addressing modes). If there is a motor- or coordinate-system-specifier (**#n** or **&n**) within the quoted string, a motor- or coordinate-system-specific command will be directed to that motor or coordinate system. If there is no specifier, a motor- or coordinate-system-specific command will be directed to the first motor or coordinate system. Any specifier within a **COMMAND** statement is not modal; it does not affect the host addressing specifications or the modal addressing of any program, including its own.

If I62=0, PMAC automatically issues a carriage-return **<CR>** character at the end of any data response to the command. If I62=1, PMAC does not issue a **<CR>** character at the end of the data response; a **SEND^M** must be used to issue a **<CR>** in this case.

Each PLC program has its own addressing mode for both motors and coordinate systems, independent of each other and independent of the host addressing modes. These are controlled by the PLC program **ADDRESS** command. This modal addressing affects commands issued from within a PLC program that do not have motor or coordinate-system specifiers. At power-up/reset, all PLC programs are addressing Motor 1 and C.S.1.

There is no modal **ADDRESS** command in motion programs. Any motor-specific or coordinate-system-specific command issued from within a motion program without a specifier is automatically addressed to Motor 1 or C.S.1, respectively.

Commands issued from within a program are placed in the command queue, to be parsed and acted upon at the appropriate time by PMAC's command interpreter, which operates in background, between other background tasks. If issued from a motion program, the command will not be interpreted before the next move or dwell command in the motion program is calculated. If issued from a PLC program, the command will not be interpreted before the end of the current scan of the PLC. This delay can make the action appear to execute out of sequence.

Because of the queuing of commands and the fact that command interpretation is a lower priority than command issuing, it is possible to overflow the queue. If there is no room

for a new command, program execution is temporarily halted until the new command can be placed on the queue.

Also, commands that generate a response to the host (including errors if I6 is not equal to 2) potentially can fill up the response queue if there is no host or the host is not prepared to read the responses. This will temporarily halt program execution until the response queue is emptied. In standalone applications, it is a good idea to set I1 to 1, disabling the serial handshake, so any responses can be sent out the serial port (the default response port) at any time, even if there is no host to receive it.

In a PLC program, it is a good idea to have at least one of the conditions that caused the command issuance to occur set false immediately. This will prevent the same command from being issued again on succeeding scans of the PLC, overflowing the command and/or response queues.

Typically in a motion program, the time between moves prevents this overflow unless there are a lot of commands and the moves take a very short time.

PMAC will not issue an acknowledging character (**<ACK>** or **<LF>**) to a valid command issued from a program. It will issue a **<BELL>** character for an invalid command issued from a program unless I6 is set to 2. It is a good idea to have I6 not set to 2 in early development so you will know when PMAC has rejected such a command. Setting I6 to 2 in the actual application can prevent program hang up from a full response queue, or from disturbing the normal host communications protocol.

If PMAC variable I64 is set to 1, any response sent to the host as a result of an internal **COMMAND** statement is preceded by a **<CTRL-B>** character, making it easier for the host computer to tell that this is an unsolicited response.

Many otherwise valid commands will be rejected when issued from a motion program. For instance, you cannot jog any motor in the coordinate system executing the program, because all these motors are considered to be running in the program, even if the program is not requesting a move of the motors at that time.

When issuing commands from a program, be sure to include all the necessary syntax (motor and/or coordinate system specifiers) in the command statement or use the **ADDRESS** command. For example, use **CMD"#4HM"** and **CMD"&1A"** instead of **CMD"HM"** and **CMD"A"**. Otherwise, motor and coordinate system commands will be sent to the most recently addressed motor and coordinate system.

Example

```
COMMAND"#1J+"
CMD"#4HM"
CMD"&1B5R"
CMD"P1"
47.5
ADDRESS#3
COMMAND"J-"
IF(M40=1 AND M41=1)
      CMD"&4R"
      M41=0
ENDIF
```

See Also

Addressing Modes, On-Line Commands (Talking To PMAC)
I-variables I1, I3, I6.
Program commands **ADDRESS**, **COMMAND^{letter}**
Writing A PLC Program

COMMAND^{letter}

Function Program Control-Character Command Issuance

Type Motion program (PROG or ROT), PLC program

Syntax
```
COMMAND^{letter}
CMD^{letter}
```
where:

- **{letter}** is a letter character from A to Z (upper or lowercase) representing the corresponding control character.

Remarks This statement causes the motion program to issue a control-character command as if it came from the host. All control-character commands are global, so there are no addressing concerns.

> ***Warning:***
>
> Do not put the up-arrow character and the letter in quotes (do not use **COMMAND"^A"**) or PMAC will attempt to issue a command with the two non-control characters **^** and **A** for this example, instead of the control character.

Commands issued from within a program are placed in the command queue, to be parsed and acted upon at the appropriate time by PMAC's command interpreter, which operates in background, between other background tasks. If issued from a motion program, the command will not be interpreted before the next move or dwell command in the motion program is calculated. If issued from a PLC program, the command will not be interpreted before the end of the current scan of the PLC. This delay can make the action appear to execute out of sequence.

Because of the queuing of commands and the fact that command interpretation is a lower priority than command issuing, it is possible to overflow the queue. If there is no room for a new command, program execution is temporarily halted until the new command can be placed on the queue.

Also, commands that generate a response to the host (including errors if I6 is not equal to 2) can fill up the response queue if there is no host or the host is not prepared to read the responses. This will temporarily halt program execution until the response queue is emptied. In standalone applications, it is a good idea to set I1 to 1, disabling the serial handshake, so any responses can be sent out the serial port (the default response port) at any time, even if there is no host to receive it.

In a PLC program, it is a good idea to have at least one of the conditions that caused the command issuance to occur set false immediately. This will prevent the same command from being issued again on succeeding scans of the PLC, overflowing the command and/or response queues. Typically in a motion program, the time between moves prevents this overflow unless there are a lot of commands and the moves take a very short time.

PMAC will not issue an acknowledging character (**<ACK>** or **<LF>**) to a valid command issued from a program. It will issue a **<BELL>** character for an invalid command issued from a program unless I6 is set to 2. It is a good idea to have I6 not set to 2 in early development so you will know when PMAC has rejected such a command. Setting I6 to 2 in the actual application can prevent program hangup from a full response queue, or

from disturbing the normal host communications protocol.

If PMAC variable I64 is set to 1, any response sent to the host as a result of an internal **COMMAND** statement is preceded by a **<CTRL-B>** character, making it easier for the host computer to tell that this is an unsolicited response.

Example **CMD^D** would disable all PLC programs (equivalent to issuing a **<CONTROL-D>** from the host).
CMD^K would kill (disable) all motors on PMAC.
CMD^A would stop all programs and moves on PMAC, also closing any loops that were open.

See Also I-variables I1, I6
On-line commands **<CONTROL-A>** to **<CONTROL-Z>**
Program command **COMMAND"{command}"**

D{data}

Function Tool Data (D-Code)

Type Motion program

Syntax **D{data}**
where:

- **{data}** is a floating-point constant or expression in the range 0.000 to 999.999, specifying the program number and the line label to jump to

Remarks PMAC interprets this statement as a **CALL 10n3.({data'}*1000)** command, where **n** is the hundreds' digit of **{data}**, and **{data'}** is the value of **{data}** without the hundred's digit (modulo 100 in mathematical terms). That is, this statement causes a jump (with return) to motion program 10n3, and the specified line label. (Programs 10n3 are usually used to implement the tool data operations as the system designer sees fit.) The value of **{data'}** can be from 0.0 to 99.999, corresponding to line labels **N0** to **N99999**.

If the specified program and/or line label do not exist, the **D** command is ignored, and the program continues as if it were not there. No error is generated.

This structure permits the implementation of customizable D-code routines for machine-tool style applications by the writing of subroutines in motion programs 10n3. Arguments can be passed to these subroutines by following the D-code with one or more sets of **{letter}{data}**, as in **CALL** and **READ** statements.

Most users will have D-codes only in the range 0-99, which permits the use of PROG 1003 only, and allows **{data'}** to equal **{data}** for direct specification of the line label.

Example **D01** jumps to **N1000** of PROG 1003
D12 jumps to **N12000** of PROG 1003
D115 jumps to **N15000** of PROG 1013

See Also Program commands **CALL{data}**, **G{data}**, **M{data}**, **T{data}**, **RETURN**

DELAY{data}

Function Delay for Specified Time

Type Motion program

Syntax **DELAY{data}**
DLY{data}
where:

- **{data}** is a floating-point constant or expression, specifying the delay time in

milliseconds.

Remarks This command causes PMAC to keep the command positions of all axes in the coordinate system constant (no movement) for the time specified in **{data}**.

There are three differences between **DELAY** and **DWELL**. First, if **DELAY** comes after a blended move, the **TA** deceleration time from the move occurs within the **DELAY** time, not before it. Second, the actual time for **DELAY** does varies with a changing time base (current %value, from whatever source), whereas **DWELL** always uses the fixed time base (%100). Third, PMAC precomputes upcoming moves (and the lines preceding them) during a **DELAY**, but it does not do so during a **DWELL**.

A **DELAY** command is equivalent to a zero-distance move of the time specified in milliseconds. As for a move, if the specified **DELAY** time is less than the acceleration time currently in force (**TA** or 2***TS**), the delay will be for the acceleration time, not the specified **DELAY** time.

Example
```
DELAY750
DELAY(Q1+100)
```

See Also Time-Base Control (Synchronizing PMAC to External Events)
I-variables I10, Ix87, Ix88
On-line command **%{constant}**
Program commands **DWELL**, **TA**, **TS**

DISABLE PLC

Function Disable PLC Program(s)

Type Motion program (PROG or ROT), PLC program

Syntax
```
DISABLE PLC {constant}[,{constant}...]
DIS PLC {constant}[,{constant}...]
DISABLE PLC {constant}[..{constant}]
DIS PLC {constant}[..{constant}]
```
where:

- **{constant}** is an integer from 0 to 31, representing the program number.

Remarks This command causes PMAC to disable (stop executing) the specified uncompiled PLC program or programs. Execution can subsequently be resumed at the top of the program with the **ENABLE PLC** command. If it is desired to restart execution at the stopped point, execution should be stopped with the **PAUSE PLC** command, and restarted with the **RESUME PLC** command.

Execution of a PLC program can only be disabled at the end of a scan, which is either the end of the program, or after executing an **ENDWHILE** statement in the program. (A PLC program can be paused in the middle of a scan, however.)

PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs.

If no buffer is open when this command is sent to PMAC, it will be executed immediately as an on-line command.

Example
```
DISABLE PLC 1
DISABLE PLC 4,5
DISABLE PLC 7..20
DIS PLC 3,8,11
DIS PLC 0..31
```

See Also

I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**, **DISABLE PLCC**, **<CONTROL-D>**
Program command **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**

DISABLE PLCC {constant}[,{constant}...]

Function Disable Compiled PLC Programs

Type Motion program (PROG or ROT), PLC program (uncompiled or compiled), except for PLC0 and PLCC0

Syntax

```
DISABLE PLCC {constant}[,{constant}...]
DISABLE PLCC {constant}[..{constant}]
DIS PLCC {constant}[,{constant}...]
DIS PLCC {constant}[..{constant}]
```

where:

- **{constant}** is an integer from 0 to 31 representing the compiled PLC number

Remarks

Warning:

This command should not be used in a foreground PLC either uncompiled PLC 0 or compiled PLCC 0 as its operation cannot be guaranteed in these programs.

This command disables the operation of the specified compiled PLC (PLCC) programs. The programs are specified by number, and can be used singly, in a list separated by commas, or in a continuous range.

I-variable I5 is a separate master control of PLC program operation. Think of the two bits of I5 as two master circuit breakers for a house, and the individual PLC and PLCC enable/disable bits as separate light switches within the house. Both the master breaker and the switch must be on for the PLC to operate. The breakers and the switches can be operated independently without affecting the setting of the others.

Example

```
DISABLE PLCC 1
DISABLE PLCC 4,5
DISABLE PLCC 7..20
DIS PLCC 3,8,11
DIS PLC 0..31
```

See Also

I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**, **DISABLE PLCC**, **<CONTROL-D>**
Program command **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**

DISPLAY [{constant}] "{message}"

Function Display Text to Display Port

Type Motion program (PROG and ROT), PLC program

Syntax

```
DISPLAY [{constant}] "{message}"
DISP [{constant}] "{message}"
```

where:

- **{constant}** is an integer value between 0 and 79 specifying the starting character number on the display; if no value is specified, 0 is used.

- **{message}** is the ASCII text string to be displayed.

Remarks This command causes PMAC to send the string contained in **{message}** to the display port (J1 connector) for the liquid-crystal or vacuum-fluorescent display (Accessory 12 or equivalent).

The optional constant value specifies the starting point for the string on the display. It has a range of 0 to 79, where 0 is upper left, 39 is upper right, 40 is lower left, and 79 is lower right.

Example
```
DISPLAY 10"Hello World"
DISP "VALUE OF P1 IS"
DISP 15, 8.3, P1
```

See Also Display Port (Connecting PMAC to the Machine);
Accessory 12 (Basic Specifications)
Program commands **DISPLAY {variable}**, **SEND"{message}"**

DISPLAY ... {variable}

Function Formatted Display of Variable Value

Type Motion program (PROG and ROT), PLC program

Syntax
```
DISPLAY {constant}, {constant}.{constant}, {variable}
DISP {constant}, {constant}.{constant}, {variable}
```
where:

- the first **{constant}** is an integer from 0 to 79 representing the starting location (character number) on the display.
- the second **{constant}** is an integer from 2 to 16 representing the total number of characters to be used to display the value (integer digits, decimal point, and fractional digits).
- the third **{constant}** is an integer from 0 to 9 (and at least two less than the second **{constant}**) representing the number of fractional digits to be displayed.
- **{variable}** is the name of the variable to be displayed.

Remarks This command causes PMAC to send a formatted string containing the value of the specified variable to the display port. The value of any I, P, Q, or M variable may be displayed with this command.

The first constant value specifies the starting point for the string on the display. It has a range of 0 to 79, where 0 is upper left, 39 is upper right, 40 is lower left, and 79 is lower right. The second constant specifies the number of characters to be used in displaying the value. It has a range of 2 to 16. The third constant specifies the number of places to the right of the decimal point. It has a range of 0 to 9, and must be at least 2 less than the number of characters. The last thing specified in the statement is the name of the variable – I, P, Q, or M.

Example
```
DISPLAY 0, 8.0, P50
DISPLAY 24, 2.0, M1
DISPLAY 40, 12.4, Q100
```

See Also Display Port (Connecting PMAC to the Machine);
Accessory 12 (Basic Specifications)
Program commands **DISPLAY"{message}"**, **COMMAND"{command}"**

DWELL

Function Dwell for Specified Time

Type Motion program (PROG and ROT)

Syntax
DWELL{data}
DWE{data}
where:

- **{data}** is a non-negative floating point constant or expression representing the dwell time in milliseconds.

Remarks This command causes the card to keep the commanded positions of all axes in the coordinate system constant for the time specified in **{data}**.

There are three differences between **DWELL** and the similar **DELAY** command. First, if the previous servo command was a blended move, there will be a **TA** time deceleration to a stop before the dwell time starts. Second, **DWELL** is not sensitive to a varying time base – it always operates in 'real time' (as defined by I10). Third, PMAC does not pre-compute upcoming moves (and the program lines before them during the **DWELL**; it waits until after it is done to start further calculations, which it performs in the time specified by I11 or I12.

Use of any **DWELL** command, even a **DWELL0**, while in external time base will cause a loss of synchronicity with the master signal.

Example
```
DWELL250
DWELL(P1+P2)
DWE0
```

See Also Dwell and Delay (Writing a Motion Program)
I-variables I10, I11, I12.
Program command **DELAY**

ELSE

Function Start False Condition Branch

Type Motion program (PROG only), PLC program

Syntax
ELSE (Motion or PLC Program)
ELSE {action} (Motion Program only)

Remarks

Warning:

With nested **IF** branches, be careful to match the **ELSE** statements to the proper **IF** statement. In a motion program, it is possible to have a single-line **IF** statement (**IF({condition}) {action}**). An **ELSE** statement on the next program line is automatically matched to this **IF** statement. Put a non-**ELSE** statement in between to make the next **ELSE** statement match a previous **IF** statement.

This statement must be matched with an **IF** statement (**ELSE** requires a preceding **IF**, but **IF** does not require a following **ELSE**). It follows the statements executed upon a true **IF** condition. It is followed by the statements to be executed upon a false **IF**

condition.

ELSE lines can take two forms (only the first of which is valid in a PLC program):

With no statement following on that line, all subsequent statements down to the next **ENDIF** statement will be executed provided that the preceding IF condition is false.

```
ELSE
     {statement}
     [{statement}
     ...]
ENDIF
```

With a statement or statements following on that line, the single statement will be executed provided that the preceding **IF** condition is false. No **ENDIF** statement should be used in this case.

```
ELSE {statement} [{statement}...]
```

Note:

This single-line **ELSE** branch form is valid only in motion programs. If this is tried in a PLC program, PMAC will put the statements on the next program line and expect an **ENDIF** to close the branch. The logic will not be as expected.

Example

This first example has multi-line true and false branches. It could be used in either a motion program or a PLC program:

```
IF (M11=1)
      P1=17
      P2=13
ELSE
      P1=13
      P2=17
ENDIF
```

This second example has a multi-line true branch, and a single-line false branch. This structure could only be used in a motion program:

```
IF (M11=0)
      X(P1)
      DWELL 1000
ELSE DWELL 500
```

This example has a single-line true branch, and a multi-line false branch. This structure could only be used in a motion program:

```
IF (SIN(P1)>0.5) Y(1000*SIN(P1))
ELSE
      P1=P1+5
      Y(1100*SIN(P1))
ENDIF
```

This example has single-line true and false branches. This structure could only be used in a motion program:

```
IF (P1 !< 5) X10
ELSE X-10
```

See Also

Program commands **IF**, **ENDIF**.

ENABLE PLC

Function Enable PLC Buffer(s)

Type Motion program (PROG and ROT), PLC program

Syntax
```
ENABLE PLC {constant}[,{constant}...]
ENA PLC {constant}[,{constant}...]
ENABLE PLC {constant}[..{constant}]
ENA PLC {constant}[..{constant}]
```
where:

- **{constant}** is an integer from 0 to 31 representing the program number.

Remarks This command causes PMAC to enable (start executing) the specified uncompiled PLC program or programs at the top of the program. Execution of the PLC program may have been stopped with the **DISABLE PLC**, **PAUSE PLC**, or **OPEN PLC** command.

PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.

If no buffer is open when this command is sent to PMAC, it will be executed immediately as an on-line command.

Example
```
ENABLE PLC 0
ENABLE PLC 1,2,5
ENABLE PLC 1..16
ENA PLC 7
```

See Also I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **<CONTROL-D>**
Program command **DISABLE PLC**

ENABLE PLCC

Function Enable Compiled PLC Program(s)

Type Motion program (PROG and ROT), PLC program (uncompiled and compiled)

Syntax
```
ENABLE PLCC {constant}[,{constant}...]
ENABLE PLCC {constant}[..{constant}]
ENA PLCC {constant}[,{constant}...]
ENA PLCC {constant}[..{constant}]
```
where:

- **{constant}** is an integer from 0 to 31 representing the compiled PLC number.

Remarks This command enables the operation of the specified compiled PLC (PLCC) buffers, provided I5 is set properly to allow their operation. The programs are specified by number, and can be used singly, in a list separated by commas, or in a continuous range.

I-variable I5 is a separate master control of PLC program operation. Think of the two bits of I5 as two master circuit breakers for a house, and the individual PLC and PLCC enable/disable bits as separate light switches within the house. Both the master breaker and the switch must be on for the PLC to operate. The breakers and the switches can be operated independently without affecting the setting of the others.

Example
```
ENABLE PLCC 0
ENABLE PLCC 1,2,5
ENABLE PLCC 1..16
ENA PLCC 7
```

See Also I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **ENABLE PLCC**, **DISABLE PLCC**, **<CONTROL-D>**
Program command **ENABLE PLC**, **DISABLE PLC**, **DISABLE PLCC**

ENDIF

Function Mark End of Conditional Block

Type Motion program (PROG only), PLC program

Syntax
```
ENDIF
ENDI
```

Remarks This statement marks the end of a conditional block of statements begun by an **IF** statement. It can close out the "true" branch, following the **IF** statement, in which case there is no false branch, or it can close out the "false" branch, following the **ELSE** statement.

When nesting conditions, it is important to match this **ENDIF** with the proper **IF** or **ELSE** statement. In a PLC program, every **IF** or **IF/ELSE** pair must take an **ENDIF**, so the **ENDIF** always matches the most recent **IF** statement that does not already have a matching **ENDIF**. In a motion program an **IF** or **ELSE** statement with action on the same line does not require an **ENDIF**, so the **ENDIF** would be matched with a previous **IF** statement.

Example
```
IF (P1>0)
      X1000
ENDIF
IF (P5=7)
      X1000
ELSE
      X2000
ENDIF
```

See Also Logical Structures (Writing a Motion Program)
Conditional Statements (Writing a PLC Program)
Program commands **IF**, **ELSE**

ENDWHILE

Function Mark End of Conditional Loop

Type Motion program (PROG only), PLC program

Syntax
```
ENDWHILE
ENDW
```

Remarks This statement marks the end of a conditional loop of statements begun by a **WHILE** statement. **WHILE** loops can be nested, so an **ENDWHILE** statement matches the most recent **WHILE** statement not already matched by a previous **ENDWHILE** statement.

In a motion program a **WHILE** statement with an action on the same line does not require a matching **ENDWHILE**.

In the execution of a PLC program, when an **ENDWHILE** statement is encountered, that scan of the PLC is ended, and PMAC goes onto other tasks (communications, other

PLCs). The next scan of this PLC will start at the matching **WHILE** statement.

In the execution of a motion program, if PMAC finds two jumps backward (toward the top) in the program while looking for the next move command, PMAC will pause execution of the program and not try to blend the moves together. It will go on to other tasks and resume execution of the motion program on a later scan. Two statements can cause such a jump back: **ENDWHILE** and **GOTO** (**RETURN** does not count).

The pertinent result is that PMAC will not blend moves when it hits two **ENDWHILE** statements (or the same **ENDWHILE** twice) between execution of move commands.

Example

```
WHILE (Q10<10)
      Q10=Q10+1
ENDWHILE
```

See Also Program commands **WHILE**, **ENDIF**

F{data}

Function Set Move Feedrate (Velocity)

Type Motion program (PROG and ROT)

Syntax **F{data}**
where:

- **{data}** is a positive floating-point constant or expression representing the vector velocity in user length units per user time units.

Remarks This statement sets the commanded velocity for upcoming **LINEAR** and **CIRCLE** mode blended moves. It will be ignored in other types of moves (**SPLINE**, **PVT**, and **RAPID**). It overrides any previous **TM** or **F** statement, and is overridden by any following **TM** or **F** statement.

The units of velocity specified in an **F** command are scaled position units (as set by the axis definition statements) per time unit (defined by "Feedrate Time Unit" I-variable for the coordinate system: Ix90).

The velocity specified here is the vector velocity of all of the feedrate axes of the coordinate system. That is, the move time is calculated as the vector distance of the feedrate axes (square root of the sum of the squares of the individual axes), divided by the feedrate value specified here. The minimum effective feedrate value will provide a move time of 2^{23} msec. The maximum effective feedrate value will provide a move time of 1 msec. Any non-feedrate axes commanded to move on the same move-command line will move at the speed necessary to finish in this same amount of time.

Note:

If the vector distance of a feedrate-specified move is so short that the computed move time (vector distance divided by feedrate) would be less than the acceleration time currently in force (**TA** or 2***TS**), the move will take the full acceleration time instead, and the axes will move more slowly than specified by the **F** command.

Axes are designated as feedrate axes with the **FRAX** command. If no **FRAX** command is used, the default feedrate axes are the X, Y, and Z axes. Any axis involved in circular interpolation is automatically a feedrate axis, regardless of whether it was specified in the latest **FRAX** command. In multi-axis systems, feedrate specification of moves is really only useful for systems with Cartesian geometries, for which these moves give a constant

velocity in the plane or in 3D space, regardless of movement direction.

> ***Note:***
>
> If only non-feedrate axes are commanded to move in a feedrate-specified move, PMAC will compute the vector distance (and the move time as zero) and will attempt to do the move in the acceleration time (TA or 2*TS).

Example

```
F100
F31.25
F(Q10)
F(SIN(P8*P9))
```

See Also

I-variables Ix87, Ix88, Ix89, Ix90
On-line commands **#{constant}->{axis definition}**, **FRAX**
Program commands **FRAX**, **LINEAR**, **CIRCLE**, **TM**, **TA**, **TS**

FRAX

Function

Specify Feedrate Axes

Type

Motion program (PROG and ROT)

Syntax

FRAX [({axis}[,{axis}...])]
where:

- **{axis}** is a character (X, Y, Z, A, B, C, U, V, W) specifying which axis is to be used in the vector feedrate calculations.

> ***Note:***
>
> No spaces are permitted in this command.

Remarks

This command specifies which axes are to be involved in the vector-feedrate (velocity) calculations for upcoming feedrate-specified (**F**) moves. PMAC calculates the time for these moves as the vector distance (square root of the sum of the squares of the axis distances) of all the feedrate axes divided by the feedrate. Any non-feedrate axes commanded on the same line will complete in the same amount of time, moving at whatever speed is necessary to cover the distance in that time.

Vector feedrate has obvious geometrical meaning only in a Cartesian system, for which it results in constant tool speed regardless of direction, but it is possible to specify for non-Cartesian systems, and for more than three axes.

> ***Note:***
>
> If only non-feedrate axes are commanded to move in a feedrate-specified move, PMAC will compute the vector distance (and the move time as zero) and will attempt to do the move in the acceleration time (TA or 2*TS).

The **FRAX** command without arguments causes all axes in the coordinate system to be feedrate axes in subsequent move commands. The **FRAX** command with arguments causes the specified axes to be feedrate axes, and all axes not specified to be non-feedrate axes, in subsequent move commands.

If no motion program buffer is open when this command is sent to PMAC, it will be executed as an on-line coordinate system command.

Example

For a three-axes cartesian system scaled in millimeters:

```
FRAX(X,Y)
```

```
INC
X30 Y40 Z10 F100
```

Vector distance is SQRT($30^2 + 40^2$) = 50 mm. At a speed of 100 mm/sec, move time (unblended) is 0.5 sec. X-axis speed is 30/0.5 = 60 mm/sec; Y-axis speed is 40/0.5 = 80 mm/sec; Z-axis speed is 10/0.5 = 20 mm/sec.

```
Z20
```

Vector distance is SQRT($0^2 + 0^2$) = 0 mm. Move time (unblended) is 0.0 sec, so Z-axis speed is limited only by acceleration parameters.

```
FRAX(X,Y,Z)
INC
X-30 Y-40 Z120 F65
```

Vector distance is SQRT($-30^2 + -40^2 + 120^2$) = 130 mm. Move time is 130/65 = 2.0 sec. X-axis speed is 30/2.0 = 15 mm/sec; Y-axis speed is 40/2.0 = 20 mm/sec; Z-axis speed is 120/2.0 = 60 mm/sec.

See Also I-variables Ix87, Ix88, Ix89, Ix90
On-line command **FRAX**
Program commands **F**, **LINEAR**, **CIRCLE**, **{axis}{data}**.

G{data}

Function Preparatory Code (G-Code)

Type Motion program

Syntax **G{data}**
where:

- **{data}** is a floating-point constant or expression in the range 0.000 to 999.999, specifying the program number and the line label to jump to

Remarks PMAC interprets this statement as a **CALL 10n0.({data'}*1000)** command, where **n** is the hundreds' digit of **{data}**, and **{data'}** is the value of **{data}** without the hundred's digit (modulo 100 in mathematical terms). That is, this statement causes a jump (with return) to motion program 10n0, and the specified line label. (Programs 10n0 are usually used to implement the preparatory codes as the system designer sees fit.) The value of **{data'}** can be from 0.0 to 99.999, corresponding to line labels **N0** to **N99999**.

If the specified program and/or line label do not exist, the **G** command is ignored, and the program continues as if it were not there. No error is generated.

This structure permits the implementation of customizable G-code routines for machine-tool style applications by the writing of subroutines in motion programs 10n0. Arguments can be passed to these subroutines by following the G-code with one or more sets of **{letter}{data}**, as in **CALL** and **READ** statements.

Most users will have G-codes only in the range 0-99, which permits the use of PROG 1000 only, and allows **{data'}** to equal **{data}** for direct specification of the line label.

Example **G01** jumps to **N1000** of PROG 1000
G12 jumps to **N12000** of PROG 1000
G115 jumps to **N15000** of PROG 1010

See Also Program commands **CALL{data}, D{data}, M{data}, T{data}, RETURN**

GOSUB

Function Unconditional Jump With Return

Type Motion program (PROG only)

Syntax **GOSUB{data}**
where:

- **{data}** is a constant or expression representing the line label to jump to.
- **{letter}** (optional) is any letter character except N or O.

Remarks This command causes the motion program execution to jump to the line label (**N** or **O**) of the same motion program specified in **{data}**, with a jump back to the commands immediately following the **GOSUB** upon encountering the next **RETURN** command.

If **{data}** is a constant, the path to the subroutine will have been linked before program run time, so the jump is very quick. If **{data}** is a variable expression, it must be evaluated at run time, and the appropriate label then searched for.

The search starts downward in the program to the end, then continues (if necessary) from the top of the program down.

A variable **GOSUB** command permits the equivalent structure to the CASE statement found in many high-level languages.

If the specified line label is not found, the **GOSUB** command will be ignored, and the program will continue as if the command had not occurred.

The **CALL** command is similar, except that it can jump to another motion program.

Example **GOSUB300** jumps to **N300** of this program, to jump back on **RETURN**.
GOSUB8743 jumps to **N8743** of this program, to jump back on **RETURN**.
GOSUB(P17) jumps to the line label of this program whose number matches the current value of P17, to jump back on **RETURN**.

See Also Writing a Motion Program
Program commands **CALL**, **GOTO**, **N**, **O**, **RETURN**

GOTO

Function Unconditional Jump Without Return

Type Motion program (PROG only)

Syntax **GOTO{data}**
where:

- **{data}** is an integer constant or expression with a value from 0 to 99,999.

Remarks This command causes the motion program execution to jump to the line label (**N** or **O**) specified in **{data}** with no jump back.

If **{data}** is a constant, the path to the label will have been linked before program run time, so the jump is very quick. If **{data}** is a variable expression, it must be evaluated at run time, and the appropriate label then searched for. The search starts downward in the program to the end, then continues (if necessary) from the top of the program down.

A variable **GOTO** command permits the equivalent structure to the CASE statement found in many high-level languages. (See Example, below.)

If the specified line label is not found, the program will stop, and the coordinate system's Run-Time-Error bit will be set.

> ***Note:***
>
> Modern philosophies of the proper structuring of computer code strongly discourage the use of **GOTO**, because of its tendency to make code undecipherable.

Example

```
GOTO750
GOTO35000
GOTO1
GOTO(50+P1)
N51 P10=50*SIN(P11)
GOTO60
N52 P10=50*COS(P11)
GOTO60
N53 P10=50*TAN(P11)
N60 X(P10)
```

See Also Writing a Motion Program;
Program commands **CALL**, **GOSUB**, **N**, **O**.

HOME

Function Programmed Homing

Type Motion program

Syntax

```
HOME {constant} [,{constant}...]
HOME {constant}..{constant} [,{constant}..{constant}...]
HM {constant} [,{constant}...]
HM {constant}..{constant} [,{constant}..{constant}...]
```

where:

- **{constant}** is an integer from one to eight representing a motor number.

Remarks This causes the specified motors to go through their homing search cycles. The motors must be specified directly by number, not the matching axis letters. Also specify which motors are to be homed. All motors specified in a single **HOME** command (e.g., **HOME1,2**) will start their homing cycles simultaneously. To home some motors sequentially, specify them in consecutive commands (e.g., **HOME1 HOME2**), even if on the same line.

Any previous moves will come to a stop before the home moves start. No other program statement will be executed until all specified motors have finished homing. Homing direction, speed, acceleration, etc., are determined by motor I-variables. If a motor is specified that is not in the coordinate system running the program, the command or portion of the command will be ignored, but an error will not be generated.

The speed of the home search move is determined by Ix23. If Ix23=0 then the programmed home command for that axis is ignored.

Note:

Unlike an on-line homing command, the motor numbers in a program homing command are specified after the word **HOME** itself, not before. In addition, an on-line homing command simply starts the homing search – it does not give any indication when the search is complete; but a program homing command automatically recognizes the end of the search, and then continues on in the program. A PLC program can only issue an on-line home command.

Example

```
HOME1                    ;These are motion program commands
HM1,2,3
HOME1..3,5..7
HM1..8
#1HOME                   ;These are on-line commands
#1HM,#2HM,#3HM
```

See Also

Homing-Search Moves (Basic Motor Moves)
On-line motor commands **HOME**, **HOMEZ**
Program command **HOMEZ**

HOMEZ

Function

Programmed Zero-Move Homing

Type

Motion program

Syntax

```
HOMEZ {constant} [,{constant}...]
HOMEZ {constant}..{constant} [,{constant}..{constant}...]
HMZ {constant} [,{constant}...]
HMZ {constant}..{constant} [,{constant}..{constant}...]
```

where:

- **{constant}** is an integer from one to eight representing a motor number.

Remarks

This commands causes the specified motors to go through pseudo-homing search cycles. In this operation, the present commanded position of the motor is made the zero position for the motor and the new commanded position for the motor.

If there is a following error and/or an axis definition offset at the time of the **HOMEZ** command, the reported position after the command will be equal to the negative of the following error plus the axis definition offset.

The motors must be specified directly by number, not the matching axis letters. Specify which motors are to be homed. All motors specified in a single **HOMEZ** command (e.g. **HOMEZ1,2**) will home simultaneously.

Note:

Unlike an on-line homing command, the motor numbers in a program homing command are specified after the word **HOMEZ** itself, not before.

Example

```
HOMEZ1                   ;These are motion program commands
HMZ1,2,3
HOMEZ1..3,5..7
HMZ1..8
#1HOMEZ                  ;These are on-line commands
#1HMZ,#2HMZ,#3HMZ
```

See Also Homing-Search Moves (Basic Motor Moves)
On-line motor command **HOME**, **HOMEZ**
Program command **HOME**

I{data}

Function I-Vector Specification for Circular Moves or Normal Vectors

Type Motion program (PROG or ROT)

Syntax **I{data}**
where:

- **{data}** is a floating-point constant or expression representing the magnitude of the I-component of the vector in scaled user axis units.

Remarks In circular moves, this specifies the component of the vector to the arc center that is parallel to the X-axis. The starting point of the vector is either the move start point (for **INC (R)** mode – default) or the XYZ-origin (for **ABS (R)** mode).

In a **NORMAL** command, this specifies the component of the normal vector to the plane of circular interpolation and tool radius compensation that is parallel to the X-axis.

Example

```
X10 Y20 I5 J5
X(2*P1) I(P1)
I33.333          specifies a full circle whose center is 33.333 units in the positive
                 X-direction from the start and end point
NORMAL I-1       specifies a vector normal to the YZ plane
```

See Also Circular Interpolation, Tool Radius Compensation (Writing a Motion Program)
On-line command **I{constant}**
Program Commands **{axis}{data}{vector}{data}**, **ABS**, **INC**, **NORMAL**, **J**, **K**, **I{constant}={expression}**

I{constant}={expression}

Function Set I-Variable Value

Type Motion program (PROG and ROT), PLC Program

Syntax **I{constant}={expression}**
where:

- **{constant}** is an integer value from 0 to 1023 representing the I-variable number.
- **{expression)** represents the value to be assigned to the specified I-variable.

Remarks This command sets the value of the specified I-variable to that of the expression on the right side of the equals sign. The assignment is done as the line is processed, which in a motion program is usually one or two moves ahead of the move actually executing at the time (because of the need to calculate ahead in the program).

> ***Note:***
>
> If the assignment of the I-variable value should be synchronous with the beginning of the next move in the program, assign an M-variable to the register of the I-variable, and use a synchronous M-variable assignment statement (**M{constant}=={expression}**).

Example

```
I130=30000
I902=1
I131=P131+1000
```

See Also How PMAC Executes a Motion Program (Writing a Motion Program)
On-line command **I{constant}={expression}**
Program commands **M{constant}={expression)**, **P{constant}={expression}**, **Q{constant}={expression}**, **M{constant}=={expression}**

IDIS{constant}

Function Incremental displacement of X, Y, and Z axes

Type Motion program (PROG and ROT)

Syntax **IDIS{constant**
where:

- **{constant}** is an integer representing the number of the first of three consecutive Q-variables to be used in the displacement vector.

Remarks This command adds to the offset values of the currently selected (with **TSEL**) transformation matrix for the coordinate system the values contained in the three Q-variables starting with the specified one.

This has the effect of renaming the current commanded X, Y, and Z axis positions (from the latest programmed move) by adding the values of these variables (Xnew=Xold+Q{constant}, Ynew=Yold+Q({constant}+1), Znew=Zold+Q({constant}+2)). This command does not cause any movement of any axes. It simply renames the present positions.

This command is similar to a **PSET** command, except that **IDIS** is incremental and does not force a stop between moves, as **PSET** does.

Example

```
X0 Y0 Z0
Q20=7.5
Q21=12.5
Q22=20
IDIS 20                    ; This makes the current position X7.5, Y12.5, Z20
IDIS 20                    ; This makes the current position X15 Y25 Z40
```

See Also Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **AROT**, **IROT**, **TINIT**

IF ({condition})

Function Conditional branch

Type Motion and PLC program

Syntax **IF ({condition})** (Valid in fixed motion (PROG) or PLC program only)
IF ({condition}) {action} [{action}...] (Valid in rotary or fixed motion program only)
where:

- **{condition}** consists of one or more sets of **{expression} {comparator} {expression}**, joined by logical operators **AND** or **OR**.
- **{action}** is a program command.

Remarks This command allows conditional branching in the program.

With an action statement or statements following on that line, it will execute those statements provided the condition is true (this syntax is valid in motion programs only).

If the condition is false, it will not execute those statements.

It will only execute any statements on a false condition if the line immediately following begins with **ELSE**. If the next line does not begin with **ELSE**, there is an implied **ENDIF** at the end of the line. When there is an **ELSE** statement on the motion-program line immediately following an **IF** statement with actions on the same line, that **ELSE** statement is automatically matched to this **IF** statement, not to any preceding **IF** statements under which this **IF** statement may be nested.

With no statement following on that line, if the condition is true, PMAC will execute all subsequent statements on following lines down to the next **ENDIF** or **ELSE** statement (this syntax is valid in motion and PLC programs). If the condition is false, it will skip to the **ENDIF** or **ELSE** statement and continue execution there.

In a rotary motion program, only the single-line version of the **IF** statement is permitted. No **ELSE** or **ENDIF** statements are allowed.

In a PLC program, compound conditions can be extended onto multiple program lines with subsequent **AND** and **OR** statements.

There is no limit on nesting of **IF** conditions and **WHILE** loops (other than total buffer size) in fixed motion and PLC programs. No nesting is allowed in rotary motion programs.

Example

```
IF (P1>10) M1=1                          ; OK in PROG & ROT, not in PLC
IF (M11=0 AND M12!=0) M2=1 M3=1          ; OK in PROG & ROT, not in PLC
                                         ; OK in PROG only, not ROT or PLC
IF (M1=0) P1=P1-1
ELSE P1=P1+1                             ; OK in PROG only, not ROT or PLC

IF (M11=0)
      P1=1000*SIN(P5)
      X(P1)
ENDIF

                                         ; OK in PLCs, not PROG or ROT
IF (P1<0 OR P2!<0)
AND (P50=1)
      P10=0
ELSE
      P10=1
ENDIF
```

See Also

Conditions (Program Computational Features)
Program commands **ELSE**, **ENDIF**, **WHILE**, **AND**, **OR**

INC

Function

Incremental Move Mode

Type

Motion program

Syntax

INC [({axis}[,{axis}...])]

where:

- **{axis}** is a letter specifying a motion axis (X, Y, Z, A, B, C, U, V, W), or the letter R specifying the arc center radial vector.

> ***Note***
>
> No spaces are permitted in this command.

Remarks The **INC** command without arguments causes all subsequent command positions in motion commands for all axes in the coordinate system running the motion program to be treated as incremental distances from the latest command point. This is known as incremental mode, as opposed to the default absolute mode.

An **INC** statement with arguments causes the specified axes to be in incremental mode, and all others stay the way they were before.

If **R** is specified as one of the axes, the I, J, and K terms of the circular move radius vector specification will be specified in incremental form (i.e. as a vector from the move start point, not from the origin). An **INC** command without any arguments does not affect this vector specification. The default radial vector specification is incremental.

If no motion program buffer is open when this command is sent to PMAC, it will be executed as an on-line coordinate system command.

Example

```
INC(A,B,C)
INC
INC(U)
INC(R)
```

See Also Circular Moves (Writing a Motion Program)
On-line commands **ABS**, **INC**
Program commands **{axis}{data}**, **{axis}{data}{vector}{data}**, **ABS**.

IROT{constant}

Function Incremental rotation/scaling of X, Y, and Z axes

Type Motion program (PROG and ROT)

Syntax **IROT{constant}**
where:

- **{constant}** is an integer representing the number of the first of nine consecutive Q-variables to be used in the rotation/scaling matrix

Remarks This command multiplies the currently selected (with **TSEL**) transformation matrix for the coordinate system by the rotation/scaling values contained in the nine Q-variables starting with the specified one. This has the effect of renaming the current commanded X, Y, and Z axis positions (from the latest programmed move) by multiplying the existing rotation/scaling matrix by the matrix containing these Q-variables, adding angles of rotation and multiplying scale factors.

The rotation and scaling is done relative to the latest rotation and scaling of the XYZ coordinate system, defined by the most recent **AROT** or **IROT** commands. The math performed is:

$$[\text{New Rot Matrix}] = [\text{Old Rot Matrix}]\ [\text{Incremental Rot Matrix}]$$
$$[\text{Xrot Yrot Zrot}]^T = [\text{New Rot Matrix}]\ [\text{Xbase Ybase Zbase}]^T$$

This command does not cause movement of any axes. It simply renames the present positions.

> ***Note:***
> When using this command to scale the coordinate system, do not use the radius center specification for circle commands. The radius does not get scaled. Use the **I, J, K** vector specification instead.

Example Create a 3x3 matrix to rotate the XY plane by 30 degrees about the origin

```
Q40=COS(30) Q41=SIN(30) Q42=0
Q43=-SIN(30)       Q44=COS(30) Q45=0
Q46=0              Q47=0              Q48=1
IROT 40                    ; Implement the change, rotating 30 degrees from current
IROT 40                    ; This rotates a further 30 degrees
```

Create a 3x3 matrix to scale the XYZ space by a factor of 3

```
Q50=3        Q51=0              Q52=0
Q53=0        Q54=3              Q55=0
Q56=0        Q57=0              Q58=3
IROT 50                    ; Implement the change, scaling up by a factor of 3
IROT 50                    ; Scale up by a further factor of 3 (total of 9x)
```

See Also Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **IDIS**, **AROT**, **TINIT**

J{data}

Function J-Vector Specification for Circular Moves

Type Motion program (PROG and ROT)

Syntax **J{data}**
where:

- **{data}** is a floating-point constant or expression representing the magnitude of the J-component of the vector in scaled user axis units.

Remarks In circular moves, this specifies the component of the vector to the arc center that is parallel to the Y-axis. The starting point of the vector is either the move start point (for **INC (R)** mode – default) or the XYZ-origin (for **ABS (R)** mode).

In a **NORMAL** command, this specifies the component of the normal vector to the plane of circular interpolation and tool radius compensation that is parallel to the Y-axis.

Example

```
X10 Y20 I5 J5
Y(2*P1) J(P1)
J33.333                    specifies a full circle whose center is 33.333 units in the
                           positive Y-direction from the start and end point
NORMAL J-1                 specifies a vector normal to the ZX plane
```

See Also Circular Interpolation, Tool Radius Compensation (Writing a Motion Program)
Motion Program Commands **{axis}{data}{vector}{data}**, **ABS**, **INC**, **NORMAL**, **I**, **K**.

K{data}

Function K-Vector Specification for Circular Moves

Type Motion program (PROG and ROT)

Syntax **K{data}**
where:

- **{data}** is a floating-point constant or expression representing the magnitude of the K-component of the vector in scaled user axis units.

Remarks In circular moves, this specifies the component of the vector to the arc center that is parallel to the Z-axis. The starting point of the vector is either the move start point (for **INC (R)** mode – default) or the XYZ-origin (for **ABS (R)** mode).

In a **NORMAL** command, this specifies the component of the normal vector to the plane of circular interpolation and tool radius compensation that is parallel to the Y-axis.

Example

```
X10 Z20 I5 K5
Z(2*P1) K(P1)
K33.333                 specifies a full circle whose center is 33.333 units in the
                        positive Z-direction from the start and end point
NORMAL K-1              specifies a vector normal to the XY plane
```

See Also Circular Interpolation, Tool Radius Compensation (Writing a Motion Program)
Motion Program Commands **{axis}{data}{vector}{data}**, **ABS**, **INC**, **NORMAL**, **I**, **J**.

LINEAR

Function Blended Linear Interpolation Move Mode

Type Motion program (PROG and ROT)

Syntax

```
LINEAR
LIN
```

Remarks The **LINEAR** command puts the program in blended linear move mode (this is the default condition on power-up/reset). Subsequent move commands in the program will be processed according to the rules of this mode. On each axis, the card attempts to reach a constant velocity that is determined by the most recent feedrate (**F**) or move time (**TM**) command.

The **LINEAR** command takes the program out of any of the other move modes (**CIRCLE**, **PVT**, **RAPID**, **SPLINE**). A command for any of these other move modes takes the program out of **LINEAR** mode.

Example

```
LINEAR ABS
CIRCLE1 X10 Y20 I5
LINEAR X10 Y0
OPEN PROG 1000 CLEAR
N1000 LINEAR RETURN
```

See Also Linear Blended Moves (Writing a Motion Program);
I-variables Ix87, Ix88, Ix89, Ix90;
Program commands **CIRCLE**, **PVT**, **RAPID**, **SPLINE**, **TA**, **TS**, **TM**, **F**, **{axis}{data}**.

M{constant}={expression}

Function Set M-Variable Value

Type Motion program (PROG and ROT)

Syntax **M{constant}={expression}**
where:

- **{constant}** is an integer constant from 0 to 1023 representing the number of the M-variable.
- **{expression}** is a mathematical expression representing the value to be assigned to this M-variable.

Remarks This command sets the value of the specified M-variable to that of the expression on the right side of the equals sign.

Note:

In a motion program, the assignment is done as the line is processed, not necessarily in order with the actual execution of the move commands on either side of it. If it is in the middle of a continuous move sequence, the assignment occurs one or two moves ahead of its apparent place in the program (because of the need to calculate ahead in the program).

If the actual assignment of the value to the variable should be synchronous with the beginning of the next move, use the synchronous M-variable assignment command **M{constant}=={expression** instead.

Example

```
M1=1
M102=$00FF
M161=P161*I108*32
M20=M20 & $0F
```

See Also How PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)
Program Commands **I{constant}=**, **P{constant}=**, **Q{constant}=**, **M{constant}==**.

M{constant}=={expression}

Function Synchronous M-Variable Value Assignment

Type Motion program (PROG and ROT)

Syntax **M{constant}=={expression}**
where:

- **{constant}** is an integer constant from 0 to 1023 representing the number of the M-variable.
- **{expression}** is a mathematical expression representing the value to be assigned to this M-variable.

Remarks This command allows the value of an M-variable to be set synchronously with the start of the next move or dwell. This is especially useful with M-variables assigned to outputs, so the output changes synchronously with beginning or end of the move. Non-synchronous calculations (with the single =) are fully executed ahead of time during previous moves.

Note:

This command may not be used with any of the thumbwheel-multiplexer forms of M-variables (TWB, TWD, TWR, TWS).

In this form, the expression on the right side is evaluated just as for a non-synchronous assignment, but the resulting value is not assigned to the specified M-variable until the start of the actual execution of the following motion command.

Note:

Remember that if this M-variable is used in further expressions before the next move in the program is started, the value assigned in this statement will not be received.

Example

```
X10
M1==1                    ; Set Output 1 at start of actual blending to next move.
X20
M60==P1+P2
```

See Also How PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)
Program Commands **I{constant}=**, **P{constant}=**, **Q{constant}=**, **M{constant}=**.

M{constant}&={expression}

Function M-Variable And-Equals Assignment

Type Motion program (PROG and ROT)

Syntax **M{constant}&={expression}**
where:

- **{constant}** is an integer constant from 0 to 1023 representing the number of the M-variable.
- **{expression}** is a mathematical expression representing the value to be 'ANDed' with this M-variable.

Remarks This command is equivalent to **M{constant}=M{constant}&{expression}**, except that the bit-by-bit **AND** and the assignment of the resulting value to the M-variable do not happen until the start of the actual execution of the following motion command. The expression itself is evaluated when the program line is encountered, as in a non-synchronous statement.

> ***Note:***
>
> This command may not be used with any of the thumbwheel-multiplexer forms of M-variables (TWB, TWD, TWR, or TWS), or with any of the double-word forms (L, D, or F).
>
> Remember that if you use this M-variable in further expressions before the next move in the program is started, you will not get the value assigned in this statement.

Example

```
M20&=$FE        ; Mask out LSB of byte M20
M346&=2         ; Clear all bits except bit 1
```

See Also How PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)
Program Commands **M{constant}=**, **M{constant}==**, **M{constant}|=**, **M{constant}^=**

M{constant}|={expression}

Function M Variable Or-Equals Assignment

Type Motion program (PROG and ROT)

Syntax **M{constant}|={expression}**
where:

- **{constant}** is an integer constant from 0 to 1023 representing the number of the M-variable;
- **{expression}** is a mathematical expression representing the value to be 'ORed' with this M-variable.

Remarks This form is equivalent to **M{constant}=M{constant}|{expression}**, except that the bit-by-bit **OR** and the assignment of the resulting value to the M-variable do not happen until the start of the following servo command. The expression itself is evaluated when the program line is encountered, as in a non-synchronous statement.

Note:

This command may not be used with any of the thumbwheel-multiplexer forms of M-variables (TWB, TWD, TWR, or TWS), or with any of the double-word forms (L, D, or F).

Remember that if you use this M-variable in further expressions before the next move in the program is started, you will not get the value assigned in this statement.

Example

```
M20|=$01          ; Set low bit of byte M20, leave other bits
M875|=$FF00       ; Set high byte, leaving low byte as is
```

See Also How PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)
Program Commands **M{constant}=**, **M{constant}==**, **M{constant}&=**, **M{constant}^=**

M{constant}^={expression}

Function M-Variable 'XOR-Equals' Assignment

Type Motion program (PROG and ROT)

Syntax **M{data}^={expression}**
where:

- **{constant}** is an integer constant from 0 to 1023 representing the number of the M-variable.
- **{expression}** is a mathematical expression representing the value to be 'XORed' with this M-variable.

Remarks This form is equivalent to **M{constant}=M{constant}^{expression}**, except that the bit-by-bit XOR and the assignment of the resulting value to the M-variable do not happen until the start of the following servo command. The expression itself is evaluated when the program line is encountered, as in a non-synchronous statement.

Note:

This command may not be used with any of the thumbwheel-multiplexer forms of M-variables (TWB, TWD, TWR, or TWS), or with any of the double-word forms (L, D, or F).

Remember that if you use this M-variable in further expressions before the next move in the program is started, you will not get the value assigned in this statement.

Example

```
M20^=$FF          ; Toggle all bits of byte M20
M99^=$80          ; Toggle bit 7 of M99, leaving other bits as is
```

See Also How PMAC Executes a Motion Program, Synchronous Variable Assignment (Writing a Motion Program)
Program Commands **M{constant}=**, **M{constant}==**, **M{constant}&=**, **M{constant}|=**

M{data}

Function Machine Code (M-Code)

Type Motion program

Syntax **M{data}**
where:

- **{data}** is a floating-point constant or expression in the range 0.000 to 999.999, specifying the program number and the line label to jump to.

Remarks PMAC interprets this statement as a **CALL 10n1.({data'}*1000)** command, where **n** is the hundreds' digit of **{data},** and **{data'}** is the value of **{data}** without the hundred's digit (modulo 100 in mathematical terms). That is, this statement causes a jump (with return) to motion program 10n1, and the specified line label. (Programs 10n1 are usually used to implement the machine codes as the system designer sees fit.) The value of **{data'}** can be from 0.0 to 99.999, corresponding to line labels **N0** to **N99999**.

If the specified program and/or line label do not exist, the **M** command is ignored, and the program continues as if it were not there. No error is generated.

This structure permits the implementation of customizable M-code routines for machine-tool style applications by the writing of subroutines in motion programs 10n1. Arguments can be passed to these subroutines by following the M-code with one or more sets of **{letter}{data}**, as in **CALL** and **READ** statements.

Typically, M-codes will be only in the range 0-99, which permits the use of PROG 1001 only, and allows **{data'}** to equal **{data}** for direct specification of the line label.

Example **M01** jumps to **N1000** of PROG 1001
M12 jumps to **N12000** of PROG 1001
M115 jumps to **N15000** of PROG 1011

See Also Program commands **CALL{data}, D{data}, M{data}, T{data}, RETURN**

MACROAUXREAD

Function Read MACRO auxiliary parameter value

Type Background PLC (no motion program, PLC0, or compiled PLC)

Syntax **MACROAUXREAD{NodeNum}{ParamNum}{Variable}**
MXR{NodeNum}{ParamNum}{Variable}
where:

- **{NodeNum}** is an integer constant from 0 to 15 specifying the slave number of the node.
- **{ParamNum}** is an integer constant from 0 to 65535 specifying the auxiliary parameter number for this node.
- **{Variable}** is the name of the PMAC variable (I, P, Q, or M) into which the parameter value is to be copied.

Remarks This command permits PMAC to read auxiliary register values from slave nodes across the MACRO ring. The command must specify the node number of the slave node, the auxiliary parameter number at this node, and the name of the PMAC variable to receive the value.

Only one auxiliary access (read or write) of a single node can be done on one command line.

In order to access the auxiliary registers of a MACRO node *n*, bit *n* of I1000 must be set to 1.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, PMAC will note an error condition. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example

```
MACROAUXREAD1,24,P1          ; Read Node 1 Parameter 24 into P1
MXR5,128,M100                ; Read Node 5 Parameter 128 into M100
```

See Also On-line commands **MACROAUX, MACROAUXREAD, MACROAUXWRITE**
Program commands **MACROAUXWRITE**

MACROAUXWRITE

Function Write MACRO auxiliary parameter value

Type Background PLC (no motion program, PLC0, or compiled PLC)

Syntax

```
MACROAUXWRITE{NodeNum}{ParamNum}{Variable}
MXW{NodeNum}{ParamNum}{Variable}
```

where:

- **{NodeNum}** is an integer constant from 0 to 15 specifying the slave number of the node.
- **{ParamNum}** is an integer constant from 2 to 253 specifying the auxiliary parameter number for this node.
- **{Variable}** is the name of the PMAC variable (I, P, Q, or M) from which the parameter value is to be copied.

Remarks This command permits PMAC to write auxiliary register values to slave nodes across the MACRO ring. The command must specify the node number of the slave node, the auxiliary parameter number at this node, and the name of the PMAC variable from which the value comes.

Only one auxiliary access (read or write) of a single node can be done on one command line.

In order to access the auxiliary registers of a MACRO node *n*, bit *n* of I1000 must be set to 1.

If the slave node returns an error message or the slave node does not respond within 32 servo cycles, PMAC will note an error condition. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

Example

```
MACROAUXWRITE1,24,P1          ; Write value of P1 to Node 1 Parameter 24
MXW5,128,M100                 ; Write value of M100 to Node 5 Parameter 128
```

See Also On-line commands **MACROAUX, MACROAUXREAD, MACROAUXWRITE**
Program commands **MACROAUXREAD**

MACROSLVREAD

Function Read (copy) Type 1 MACRO auxiliary parameter value

Type Uncompiled PLC 1 – 31 only

Syntax
```
MACROSLVREAD{node #},{slave variable},{PMAC variable}
MSR{node #},{slave variable},{PMAC variable}
```
where:

- **{node #}** is a constant in the range 0 to 15 representing the number of the node on the PMAC matching the slave node to be accessed.
- **{slave variable}** is the name of the variable on the slave station whose value is to be reported.
- **{PMAC variable}** is the name of the variable on the PMAC into which the value of the slave station variable is to be copied.

Remarks This command causes PMAC to copy the value of the specified variable of the MACRO slave station matching the specified node number on the PMAC to the specified PMAC variable, using the MACRO Type 1 master-to-slave auxiliary protocol.

The variable on the PMAC can be any of the I, P, Q, or M-variable on the card.

In order for the PMAC to be able to execute this command, the following conditions must be true:

- The PMAC must be set up as a master or the synchronizing ring master (I995= $xx90 or $xx30).
- The node 15 auxiliary register copy function must be disabled (I1000 bit 15 = 0).
- Node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I1003 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

If this command is issued to a PMAC when no buffer is open, it will be executed as an on-line command.

Example
```
MSR0,MI910,P1     ; Copies value of slave Node 0 variable MI910 into PMAC variable P1
MSR1,MI997,M10    ; Copies value of slave Node 1 variable MI997 into PMAC variable M10
```

MACROSLVWRITE

Function Write (copy) Type 1 MACRO auxiliary parameter value

Type Uncompiled PLC 1 – 31 only

Syntax
```
MACROSLVWRITE{node #},{slave variable},{PMAC variable}
MSW{node #},{slave variable},{PMAC variable}
```
where:

- **{node #}** is a constant in the range 0 to 15 representing the number of the node on the PMAC matching the slave node to be accessed.
- **{slave variable}** is the name of the MI-variable or C-command on the slave station whose value is to be set.
- **{PMAC variable}** is the name of the variable on the PMAC from which the value of the slave station variable is to be copied.

Remarks This command causes PMAC to copy the value of the specified variable on PMAC to the specified variable of the MACRO slave station matching the specified node number on the PMAC, using the MACRO Type 1 master-to-slave auxiliary protocol.

The variable on the PMAC can be any of the I, P, Q, or M-variables on the card.

In order for the PMAC to be able to execute this command, the following conditions must be true:

- The PMAC must be set up as a master or the synchronizing ring master (I995= $xx90 or $xx30).
- The node 15 auxiliary register copy function must be disabled (I1000 bit 15 = 0).
- Node 15 must not be used for any other function.

If the slave node returns an error message or it does not respond within I1003 servo cycles, PMAC will report ERR008. Bit 5 of global status register X:$0003 is set to report such a MACRO auxiliary communications error. Register X:$0798 holds the error value. It is set to $010000 for a timeout error, or $*xxxx*FE if the slave node reports an error, where *xxxx* is the 16-bit error code reported by the slave node.

If this command is issued to a PMAC when no buffer is open, it will be executed as an on-line command.

Example

```
MSW0,MI910,P35        ; Copies value of PMAC P35 into MACRO station
                      ; node 0 variable MI910
MSW4,C4,P0            ; Causes MACRO station with active node 4 to execute
                      ; Command #4, saving its setup variable values to
                      ; non-volatile memory (P0 is a dummy variable here)
```

N{constant}

Function Program Line Label

Type Motion program (PROG and ROT)

Syntax **N{constant}**
where:

- **{constant}** is an integer from 0 to 262,143 (2^{18}-1).

Remarks This is a label for a line in the program that allows the flow of execution to jump to that line with a **GOTO**, **GOSUB**, **CALL**, **G**, **M**, **T**, or **D** statement or a **B** command.

A line only needs a label in order to jump to that line. Line labels do not have to be in any sort of numerical order. The label must be at the beginning of a line. Remember that each location label takes up space in PMAC memory.

> ***Note:***
>
> There is always an implied **N0** at the beginning of every motion program. Putting an explicit **N0** at the beginning may be useful for people reading the program.

Example

```
N1
N65537 X1000
```

See Also Subroutines and Subprograms (Writing a Motion Program)
On-line command **B{constant}**
Program commands **O{constant}**, **GOTO**, **GOSUB**, **CALL**, **G**, **M**, **T**, **D**.

NORMAL

Function Define Normal Vector to Plane of Circular Interpolation and Cutter Radius Compensation

Type Motion program (PROG and ROT)

Syntax
```
NORMAL {vector}{data} [{vector}{data}...]
NRM {vector}{data} [{vector}{data}...]
```
where:

- **{vector}** is one of the letters I, J, and K, representing components of the total vector parallel to the X, Y, and Z axes, respectively.
- **{data}** is a constant or expression representing the magnitude of the particular vector component.

Remarks This statement defines the orientation of the plane in XYZ-space in which circular interpolation and cutter radius compensation will take place by setting the normal (perpendicular) vector to that plane.

The vector components that can be specified are I (X-axis direction), J (Y-axis direction), and K (Z-axis direction). The ratio of the component magnitudes determines the orientation of the normal vector, and therefore, of the plane. The length of this vector does not matter – it does not have to be a unit vector.

The direction sense of the vector does matter, because it defines the clockwise sense of an arc move, and the sense of cutter-compensation offset. PMAC uses a right-hand rule; that is, in a right-handed coordinate system (I x J = K), if your right thumb points in the direction of the norma vector specified here, your right fingers will curl in the direction of a clockwise arc in the circular plane, and in the direction of offset-right from direction of movement in the compensation plane.

Example The standard settings to produce circles in the principal planes will therefore be:

```
NORMAL K-1          ; XY plane – equivalent to G17
NORMAL J-1          ; ZX plane – equivalent to G18
NORMAL I-1          ; YZ plane – equivalent to G19
```

By using more than one vector component, a circular plane skewed from the principal planes can be defined:

```
NORMAL I0.866 J0.500
NORMAL J25 K-25
NORMAL J(-SIN(Q1)) K(-COS(Q1))
NORMAL I(P101) J(P201) K(301)
```

See Also Circular Blended Moves, Cutter Radius Compensation (Writing a Motion Program)
Cartesian Axes (Setting Up a Coordinate System)
Program Commands **CIRCLE1**, **CIRCLE2**, **CC0**, **CC1**, **CC2**

O{constant}

Function Alternate Line Label

Type Motion program (PROG and ROT)

Syntax
```
O{constant}
```
where:

- **{constant}** is an integer from 0 to 262,143 (2^{18}-1).

Remarks This is an alternate form of label in the motion program. It allows the flow of execution to jump to that line with a **GOTO**, **GOSUB**, **CALL**, **G**, **M**, **T**, or **D** statement or a **B** command. PMAC will store and report this as an **N{constant** statement, but **O** labels are legal to send to the program buffer. (**N10** and **O10** are identical labels to PMAC.)

A line only needs a label in order to jump to that line. Line labels do not have to be in any sort of numerical order. The label must be at the beginning of a line. Remember that each location label takes up space in PMAC memory.

Example

```
O1
O65537 X1000
```

See Also

Subroutines and Subprograms (Writing a Motion Program)
On-line command **B{constant}**
Program commands **O{constant}**, **GOTO**, **GOSUB**, **CALL**, **G**, **M**, **T**, **D**.

OR({condition})

Function Conditional OR

Type PLC program

Syntax **OR ({condition})**

Remarks This statement forms part of an extended compound condition to be evaluated in a PLC program. It must immediately follow an **IF**, **WHILE**, **AND**, or **OR** statement. This **OR** is a boolean operator logically combining the condition on its line with the condition on the program line above.

It takes lower precedence than operators within a compound condition on a single line (those within parentheses), and also lower precedence than an **AND** operator that starts a line. (**OR**s operate on groups of **AND**ed conditions.)

In motion programs, there can be compound conditions within one program line, but not across multiple program lines, so this statement is not permitted in motion programs.

Note:

This logical **OR**, which acts on conditions, should not be confused with the bit-by-bit **|** (vertical bar) or-operator, which operates on *values*.

Example

```
IF (M11=1)                          ; This branch increments P1 every cycle that
AND (M12=0)                         ; inputs M11 and M12 are different, and decrements
OR (M11=0)                          ; them every cycle that they are the same.
AND (M12=1)
   P1=P1+1
ELSE
   P1=P1-1
ENDIF

IF (M11=1 AND M12=0)                ; This does the same as above
OR (M11=0 AND M12=1)
   P1=P1+1
ELSE
   P1=P1-1
ENDIF
```

See Also

Conditions (Program Computational Features)
Writing a PLC Program
Program commands **IF**, **WHILE**, **AND**

P{constant}={expression}

Function Set P-Variable Value

Type Motion program (PROG and ROT)

Syntax **P{constant}={expression}**
where:

- **{constant}** is an integer constant from 0 to 1023 representing the P-variable number.
- **{expression}** represents the value to be assigned to this P-variable.

Remarks This command sets the value of the specified P-variable to that of the expression on the right side of the equals sign. The assignment is done as the line is processed, which in a motion program is usually one or two moves ahead of the move actually executing at the time (because of the need to calculate ahead in the program).

Example
```
P1=0
P746=P20+P40
P893=SIN(Q100)-0.5
```

See Also How PMAC Executes a Motion Program (Writing a Motion Program)
On-line command **P{constant}={expression}**
Program commands **I{constant}={expression}**, **M{constant}={expression}**, **Q{constant}={expression}**.

PAUSE PLC

Function Pause execution of PLC program(s)

Type Motion program (PROG or ROT), PLC program

Syntax
```
PAUSE PLC {constant}[,{constant}...]
PAU PLC {constant}[,{constant}...]
PAUSE PLC {constant}[..{constant}]
PAU PLC {constant}[..{constant}]
```

Remarks This command causes PMAC to stop execution of the specified uncompiled PLC program or programs, with the capability to restart execution at this point (not necessarily at the top) with a **RESUME PLC** command. Execution can also be restarted at the top of the program with the **ENABLE PLC** command.

If the PLC program is paused from within that PLC, execution is stopped immediately after the **PAUSE PLC** command.

If the PLC program is paused while it is not in the middle of a scan, which is always the case if it is paused from another background PLC, it will obviously be paused at the end of a scan – after an **ENDWHILE** or after the last line.

If the PLC program is paused while it has been interrupted in the middle of a scan (for example, from a motion program), its execution will resume after the interrupt and continue until after it executes any of the following:

- Any **ENABLE PLC**, **DISABLE PLC**, **PAUSE PLC**, or **RESUME PLC** command
- An **ENDWHILE** command
- The last line of the program

Execution will be paused at this point.

PLC programs are specified by number, and may be specified in a command singularly, in a list (separated by commas), or in a range of consecutively numbered programs.

If no buffer is open when this command is sent to PMAC, it will be executed immediately as an on-line command.

Example

```
PAUSE PLC 1
PAUSE PLC 4,5
PAUSE PLC 7..20
PAU PLC 3,8,11
PAU PLC 0..31
```

See Also

I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **<CONTROL-D>**, **PAUSE PLC**, **RESUME PLC**, **LIST PLC**
Program command **ENABLE PLC**, **DISABLE PLC**, **RESUME PLC**

PRELUDE

Function Specify automatic subroutine call function

Type Motion program

Syntax

```
PRELUDE1{command}
PRELUDE0
```

where:

- **{command}** is a subprogram call from the set **CALL{constant}, G{constant}, M{constant}, T{constant}, D{constant}**.

Remarks

The **PRELUDE1** command permits automatic insertion of a subprogram call before each subsequent motion command (e.g., **X10Y10**) or other letter-number command (e.g., **L10**) other than a line label in the motion program. The action taken is equivalent to inserting the call into the program text before each subsequent motion command or letter-number command.

The subprogram call to be performed can be specified in the **PRELUDE1** command either as a **CALL** command, or as a **G**, **M**, **T**, or **D** code. The value following the **CALL** or code letter must be a constant. It cannot be a variable or expression. It does not have to be an integer. If the routine called in the subprogram starts with a **READ** statement, the motion or letter-number command itself can become arguments for the subprogram call. Any motion command within a **PRELUDE1** subroutine or subprogram call is executed directly as a motion command, without an automatic subroutine call in front of it.

PMAC will only execute the **PRELUDE1** function if the motion or letter-number command is found at the beginning of a program line or immediately after the line label. If another type of command occurs earlier on the program line, no **PRELUDE1** function will be executed before the motion or letter-number command. If the command is on a line that is already in a subroutine or subprogram reached by a **CALL** or **GOSUB** command, no **PRELUDE1** function will be executed.

Each **PRELUDE1** command supersedes the previous **PRELUDE1** command. It is not possible to nest automatic **PRELUDE1** calls, but an automatic **PRELUDE1** call can be nested within explicit subroutine and subprogram calls.

PRELUDE0 disables any automatic subroutine call.

Example

```
PRELUDE1 CALL10          ; Insert a CALL10 before subsequent moves
X10 Y20                  ; Implicit CALL10 before this move
X20 Y30                  ; Implicit CALL10 before this move
...
OPEN PROG 10 CLEAR       ; Subprogram
Z-1                      ; Move down
DWELL 500                ; Hold position
Z1                       ; Move up
RETURN
...
G71 X7 Y15 P5            ; G71 calls PROG 1000 N71000
X8 Y16 P5                ; With PRELUDE, G71 is implied (modal)
X9 Y15 P8                ; With PRELUDE, G71 is implied (modal)
G70                      ; Stop modal canned cycles
...
OPEN PROG 1000
...
N70000                   ; G70 subroutine
PRELUDE0                 ; Stop PRELUDE calls
RETURN
N71000                   ; G71 subroutine
PRELUDE1 G71.1           ; Make G71 modal by using PRELUDE
RETURN
N71100                   ; G71.1 routine is what executes G71 modally
READ(X,Y,P)              ; Read values associated with X, Y, and P
                         {action based on parameters}
RETURN
```

See Also

Subroutines and Subprograms (Writing a Motion Program)
Program commands **CALL**, **GOSUB**, **READ**, **G**, **M**, **T**, **D**

PSET

Function

Redefine current axis positions (Position SET)

Type

Motion program

Syntax

PSET{axis}{data} [{axis}{data}...]
where:

- **{axis}** is the character specifying which axis (X, Y, Z, A, B, C, U, V, W).
- **{data}** is a constant or an expression representing the new value for this axis position.

Remarks

This command allows the user to re-define the value of an axis position in the middle of the program. It is equivalent to the RS-274 G-Code **G92**. No move is made on any axis as a result of this command – the value of the present commanded position for the axis is merely set to the specified value.

Internally, this command changes the value of the position bias register for each motor attached to an axis named in the command. This register holds the difference between the axis zero point and the motor zero (home) point.

This command automatically forces a temporary pause in the motion of the axes; no moves are blended "through" a **PSET** command. For more powerful and flexible offsets that can be done on the fly (X, Y, and Z axes only), refer to the matrix manipulation commands such as **ADIS** and **IDIS**.

Example

```
X10Y20
PSET X0 Y0                      ; Call this position (0,0)

N92000 READ(X,Y,Z)              ; To implement G92 in PROG 1000
PSET X(Q124)Y(Q125)Z(Q126)      ; Equivalent of G92 X..Y..Z.
```

See Also

Axes (Setting Up a Coordinate System)
On-line command **{axis}={constant}**
Program commands **ADIS**, **AROT**, **IDIS**, **IROT**
Suggested M-variable definitions Mx64
Memory map registers D:$0813, D:$08D3, etc.

PVT{data}

Function

Set Position-Velocity-Time mode

Type

Motion program (PROG and ROT)

Syntax

PVT{data}

where:

- **{data}** is a positive constant or expression representing the time of a segment in milliseconds (PMAC will round this value to the nearest integer in actual use).

Remarks

This command puts the motion program into Position-Velocity-Time move mode, and specifies the time for each segment of the move. In this mode, each move segment in the program must specify the ending position and velocity for the axis. Taking the starting position and velocity (from the previous segment), the ending position and velocity, and the segment time, PMAC computes the unique cubic position profile (parabolic velocity profile) to meet these constraints.

The segment time in a sequence of moves can be changed on the fly, either with another **PVT** command, or with a **TA** command. **TS**, **TM**, and **F** settings are irrelevant in this mode.

The **PVT** command takes the program out of any of the other move modes (**LINEAR**, **CIRCLE**, **SPLINE**, **RAPID**), and any of the other move mode commands takes the program out of **PVT** move mode.

Refer to the Writing a Motion Program section of this manual for more details.

Example

```
INC                     ; incremental mode, specify moves by distance
PVT200                  ; enter this mode – move time 200ms
X100:1500               ; cover 100 units ending at 1500 units/sec
X500:3000               ; cover 500 units ending at 3000 units/sec
X500:1500               ; cover 500 units ending at 1500 units/sec
X100:0                  ; cover 100 units ending at 0 units/sec
PVT(P37)
```

See Also

Position-Velocity-Time Mode Moves (Writing a Motion Program)
Program commands **{axis}{data}:{data}...**, **TA**, **LINEAR**, **CIRCLEn**, **RAPID**, **SPLINE1**.

Q{constant}={expression}

Function Set Q-Variable Value

Type Motion program (PROG and ROT); PLC program

Syntax **Q{constant}={expression}**
where:

- **{constant}** is an integer value from 0 to 1023 representing the Q-variable number
- **{expression}** represents the value to be assigned to the specified Q-variable.

Remarks This command sets the value of the specified Q-variable to that of the expression on the right side of the equals sign. The assignment is done as the line is processed, which in a motion program performing a continuous move sequence is usually one or two moves ahead of the move actually executing at the time (because of the need to calculate ahead in the program).

Because each coordinate system has its own set of Q-variables, it is important to know which coordinate system's Q-variable is affected by this command.

When executed from inside a motion program, this command affects the specified Q-variable of the coordinate system running the motion program.

When executed from inside a PLC program, this command affects the specified Q-variable of the coordinate system specified by the most recent **ADDRESS** command executed inside that PLC program. If there has been no **ADDRESS** command executed since power-on/reset, it affects the Q-variable of Coordinate System 1.

Example
```
Q1=3
Q99=2.71828
```

```
Q124=P100+ATAN(Q120)
```

See Also Q-Variables (Program Computational Features)
On-line command **Q{constant}={expression}**
Program commands **ADDRESS**, **I{constant}={expression}**, **M{constant}={expression}**, **P{constant}={expression}**

R{data}

Function Set Circle Radius

Type Motion program (PROG or ROT)

Syntax **R{data}**
where:

- **{data}** is a constant or expression representing the radius of the arc move specified in user length units.

Remarks This partial command defines the magnitude of the radius for the circular move specified on that command line. It does not affect the moves on any other command lines.. (If there is no R radius specification and no IJK vector specification on a move command line, the move will be done linearly, even if the program is in **CIRCLE** mode.)

If the radius value specified in **{data}** is greater than zero, the circular move to the specified end point will describe an arc of less than or equal to 180° with a radial length of the specified value. If the radius value specified in **{data}** is less than zero, the circular move to the specified end point will describe an arc of greater than or equal to 180° with a radial length equal to the absolute value of **{data}**.

Note:

If you use the **AROT** or **IROT** commands to scale the coordinate system, do not use the radius center specification for circle commands. The radius does not get scaled. Use the **I, J, K** vector specification instead.

If the distance from the start point to the end point is more than twice the magnitude specified in **{data}**, there is no circular arc move possible. If the distance is greater than twice **{data}** by an amount less than Ix96 (expressed in user length units), PMAC will execute a spiral to the end point. If the distance is greater by more than Ix96, PMAC will stop the program with a run-time error.

Example

```
RAPID X0 Y0                    ; Move to origin
CIRCLE1                        ; Clockwise circle mode
X10 Y10 R10                    ; Quarter circle to (10, 10)
X0 Y0 R-10                     ; Three-quarters circle back to (0, 0)
X(P101) R(P101/2)              ; Half circle to (P101, 0)
```

See Also

Circular Blended Moves (Writing a Motion Program);
I-variables I13, Ix96
Program commands **CIRCLE1**, **CIRCLE2**, **{axis}{data}{vector}{data}**

RAPID

Function

Set Rapid Traverse Mode

Type

Motion program (PROG and ROT)

Syntax

```
RAPID
RPD
```

Remarks

This command puts the program into a mode in which all motors defined to the commanded axes move to their destination points in jog-style moves. This mode is intended to create the minimum-time move from one point to another. Successive moves are not blended together in this mode, and the different motors do not necessarily all reach their end points at the same time

The accelerations and decelerations in this mode are controlled by motor jog-acceleration I-variables Ix19, Ix20, and Ix21. If global I-variable I50 is set to 0, the velocities in this mode are controlled by the motor jog speed I-variables Ix22. If I50 is set to 1, they are controlled by the motor maximum speed I-variables Ix16. Only the motor with the greatest distance-to-speed ratio for the move actually moves at this speed. All other motors are slowed from the specified speed to complete the move in approximately the same time, so that the move is nearly linear.

The **RAPID** command takes the program out of any of the other move modes (**LINEAR**, **CIRCLE**, **PVT**, **SPLINE**). Any of the other move-mode commands takes the program out of **RAPID** mode.

Example

```
RAPID X10 Y20                  ; Move quickly to starting cut position
M1=1                           ; Turn on cutter
LINEAR X12 Y25 F2              ; Start cutting moves
...
M1=0                           ; Turn off cutter
RAPID X0 Y                     ; Move quickly back to home position
```

See Also Rapid Mode Moves (Writing a Motion Program)
I-variables I50, Ix16, Ix19, Ix22
Program commands **LINEAR**, **CIRCLE**, **PVT**, **SPLINE**

READ

Function Read Arguments for Subroutine

Type Motion program (PROG only)

Syntax **READ({letter},[{letter}...])**
where:

- **{letter}** is any letter of the English alphabet, except **N** or **O**, representing the letter on the calling program line whose following value is to be read into a variable.

> ***Note:***
> No space is allowed between **READ** and the left parenthesis.

Remarks This statement allows a subprogram or subroutine to take arguments from the calling routine. It looks at the remainder of the line calling this routine (**CALL**, **G**, **M**, **T**, **D**), takes the values following the specified letters and puts them into particular Q-variables for the coordinate system. For the Nth letter of the alphabet, the value is put in Q(100+N).

It scans the calling line until it sees a letter that is not in the list of letters to **READ**, or until the end of the calling line. Each letter value successfully "read" into a Q-variable causes a bit to be set in Q100, noting that it was read (bit N-1 for the Nth letter of the alphabet). For any letter not successfully read in the most recent **READ** command, the corresponding bit of Q100 is set to zero.

The Q-variable and flag bit of Q100 associated with each letter are shown in the following table:

Letter	Target Variable	Q100 Bit	Bit Value Decimal	Bit Value Hex
A	Q101	0	1	$01
B	Q102	1	2	$02
C	Q103	2	4	$04
D	Q104	3	8	$08
E	Q105	4	16	$10
F	Q106	5	32	$20
G	Q107	6	64	$40
H	Q108	7	128	$80
I	Q109	8	256	$100
J	Q110	9	512	$200
K	Q111	10	1,024	$400
L	Q112	11	2,048	$800
M	Q113	12	4,096	$1000
N*	Q114*	13*	8,192*	$2000*
O*	Q115*	14*	16,384*	$4000*
P	Q116	15	32,768	$8000
Q	Q117	16	65,536	$10000
R	Q118	17	131,072	$20000
S	Q119	18	262,144	$40000
T	Q120	19	524,288	$80000
U	Q121	20	1,048,57	$100000

V	Q122	21	2,097,15	$200000
W	Q123	22	4,194,304	$400000
X	Q124	23	8,388,608	$800000
Y	Q125	24	16,777,216	$1000000
Z	Q126	25	33,554,432	$2000000
*Cannot be used				

Any letter may be **READ** except **N** or **O**, which are reserved for line labels (and should only be at the beginning of a line). If a letter value is read from the calling line, the normal function of the letter (e.g., an axis move) is overridden, so that letter serves merely to pass a parameter to the subroutine.

If there are remaining letter values on the calling line that are not read, those will be executed according to their normal function after the return from the subroutine.

If the **READ** function encounters a letter in the calling line that is not in the list of letters to be read, the reading action stops, even if there are other letters from the list still to be read on the calling line. For example, if the calling line were **CALL100 X10 Y20 Z30**, and PROG 100 started with a **READ(X,Z)**, the X-value would be read successfully, but not the Z-value.

Example

In standard machine tool code, a two-second **DWELL** would be commanded in the program as a **G04 X2000**, for instance. In PMAC, a **G04** is interpreted as a **CALL** to label **N04000** of PROG 1000, so to implement this function properly, PROG 1000 would contain the following code:

```
N04000 READ(X)
DWELL (Q124)
RETURN
```

Also, in standard machine tool code, the value assigned to the current position of the axis may be changed with the **G92** code, followed by the letters and the new assigned values of any axes (e.g. **G92 X20 Y30**). It is important only to assign new values to axes specified in this particular **G92** command, so the PMAC subroutine implementing **G92** with the **PSET** command must check to see if that particular axis is specified:

```
N92000 READ(X,Y,Z)
IF (Q100 & $800000 > 0) PSET X(Q124)
IF (Q100 & $1000000 > 0) PSET Y(Q125)
IF (Q100 & $2000000 > 0) PSET Z(Q126)
RETURN
```

See Also

Subroutines and Subprograms (Writing a Motion Program)
Program commands **CALL**, **GOSUB**, **G**, **M**, **T**, **D**

RESUME PLC

Function

Resume execution of PLC programs(s)

Type

Motion program (PROG and ROT), PLC program

Syntax

```
RESUME PLC {constant}[,{constant}...]
RES PLC {constant}[,{constant}...]
RESUME PLC{constant}[..{constant}]
RES PLC {constant}[..{constant}]
```

where:

- **{constant}** is an integer from 0 to 31, representing the program number.

Remarks

This command causes PMAC to resume execution of the specified uncompiled PLC program or programs at the point where execution was suspended with the **PAUSE PLC** command, which is not necessarily at the top of the program.

The **RESUME PLC** command cannot be used to restart execution of a PLC program that has been stopped with a **DISABLE PLC** command. However, after a PLC has been stopped with a **DISABLE PLC** command, if a **PAUSE PLC** command is then given for that PLC, then a **RESUME PLC** command can be given to start operation at the point at which it has been stopped.

PLC programs are specified by number, and may be used singularly in this command, in a list (separated by commas), or in a range of consecutively numbered programs.

If no buffer is open when this command is sent to PMAC, it will be executed immediately as an on-line command.

Example

```
RESUME PLC 0
RESUME PLC 1,2,5
RESUME PLC 1..16
RES PLC 7
```

See Also

I-variable I5
On-line commands **ENABLE PLC**, **DISABLE PLC**, **<CONTROL-D>**, **PAUSE PLC**, **RESUME PLC**
Program commands **ENABLE PLC**, **DISABLE PLC**, **PAUSE PLC**

RETURN

Function Return From Subroutine Jump/End Main Program

Type Motion program (PROG only)

Syntax

```
RETURN
RET
```

Remarks

The **RETURN** command tells the motion program to jump back to the routine that called the execution of this routine. If this routine was started from an on-line command (Run), program execution stops and the program pointer is reset to the top of this motion program. Control is returned to the PMAC "operating system".

If this routine was started from a **GOSUB**, **CALL**, **G**, **M**, **T**, or **D** command in a motion program, program execution jumps back to the command immediately following the calling command.

When the **CLOSE** command is sent to end the entry into a motion program buffer, PMAC automatically appends a **RETURN** command to the end of that program. When the **OPEN** command is sent to an existing motion program buffer, the final **RETURN** command is automatically removed.

Example

```
OPEN PROG 1 CLEAR
X20 F10
X0
CLOSE                          ; PMAC places a RETURN here
OPEN PROG 1000 CLEAR
N0 RAPID RETURN                ; Execution jumps back after one-line routine
N1000 LINEAR RETURN            ; Ditto
N2000 CIRCLE1 RETURN           ; Ditto
...
CLOSE                          ; PMAC places a RETURN here
```

See Also

Subroutines and Subprograms (Writing a Motion Program)
On-line commands **OPEN**, **CLOSE**
Program commands **CALL**, **GOSUB**, **G**, **M**, **T**, **D**

S{data}

Function Spindle data command

Type Motion program (PROG and ROT)

Syntax **S{data}**
where:

- **{data}** is a constant or expression representing the value to be passed to the storage variable for later use.

Remarks This command causes the value in **{data}** to be loaded in variable Q127 for the coordinate system executing the motion program. It takes no other action. It is intended to pass spindle speed data in machine tool programs. The algorithms that actually control the spindle would then use Q127 in their routines (e.g., to set jog speed, or voltage output).

> ***Note:***
>
> This command is distinct from **S{data}** information passed as part of a subroutine call through a **READ(S)** command. In this form, the value is placed in Q119 for the coordinate system.

Example

```
S1800                ; This puts a value of 1800 in Q127
S(P1)                ; This puts the value of P1 in Q127
G96 S50              ; Here the S-term is an argument in the G-code call
(PROG 1000)          ; This is the subroutine that executes G96
N96000 READ(S)       ; This puts a value of 50 in Q119
```

See Also Q-variables (Program Computational Features)
Implementing a Machine-Tool Style Program (Writing a Motion Program)
Motion program command **READ**
Example program SPINDLE.PMC

SEND

Function Cause PMAC to Send Message

Type Motion program (PROG and ROT); PLC program

Syntax

```
SEND"{message}"
SENDS"{message}"
SENDP"{message}"
```

Remarks This command causes PMAC to send the specified message out of one of PMAC's communications ports. This is particularly useful in the debugging of applications. It can also be used to prompt an operator, or to notify the host computer of certain conditions.

If I62=0, PMAC automatically issues a carriage-return (**<CR>**) character at the end of the message. If I62=1, PMAC does not issue a **<CR>** character at the end of the message. A **SEND^M** must be used to issue a **<CR>** in this case.

Note:

If there is no host on the port to which the message is sent, or the host is not ready to read the message, the message is left in the queue. If several messages back up in the queue this way, the program issuing the messages will halt execution until the messages are read. This is a common mistake when the **SEND** command is used outside of an Edge-Triggered condition in a PLC program. See Writing A PLC Program in Chapter 3 for more details.

On the serial port, it is possible to send messages to a non-existent host by disabling the port handshaking with I1=1.

SEND transmits over the active communications response port, whether serial, parallel host port (PC-Bus or STD-Bus), VME-Bus port, or ASCII DPRAM buffer.

SENDS always transmits over the serial port regardless of which port is the current active response port.

SENDP always transmits over the parallel host port (PC-Bus or STD-Bus), regardless of which port is the current active response port.

There is no **SENDV** command for the VME bus exclusively. The **SEND** command must be used with the VME port as the active response port.

When PMAC powers up or resets, the active response port is the serial port. When any command is received over a bus port, the active response port becomes the bus port. PMAC must then receive a **<CONTROL-Z>** command to cause the response port to revert back to the serial port.

Note:

If a program, particularly a PLC, sends messages immediately on power-up/reset, it can confuse a host-computer program (such as the PMAC Executive Program) that is trying to "find" PMAC by querying it and looking for a particular response.

It is possible, particularly in PLC programs, to order the sending of messages faster than the port can handle them. Usually this will happen if the same **SEND** command is executed through every scan in the PLC. For this reason, it is good practice to have at least one of the conditions that causes the **SEND** command to execute to be set false immediately to prevent execution of this **SEND** command on subsequent scans of the PLC.

Note:

To cause PMAC to send the value of a variable, use the **COMMAND** statement instead, specifying the name of the variable in quotes (e.g. **CMD"P1"**)

Example

```
SEND"Motion Program Started"
SENDS"DONE"
SENDP"Spindle Command Given"

IF (M188=1)                          ; C.S.1 Warning Following Error Bit set?
  IF (P188=0)                        ; But not set last scan? (P188 follows M188)
    SEND"Excessive Following Error"  ; Notify operator
```

```
    P188=1                          ; To prevent repetition of message
  ENDIF
ELSE                                ; F.E. Bit not set
  P188=0                            ; To prepare for next time
ENDIF

SEND"THE VALUE OF P7 IS:"           ; PMAC to send the message string
CMD"P7"                             ; PMAC to return the value of P7
```

See Also I-variables I1, I62
Program commands **COMMAND**, **DISPLAY**, **SEND^{letter}**
Writing A PLC Program

SEND^{letter}

Function Cause PMAC to Send Control Character

Type Motion program (PROG and ROT); PLC program

Syntax

```
SEND^{letter}
SENDS^{letter}
SENDP^{letter}
```

where:

- **{letter}** is one of the characters in the following set: **@ABC...XYZ[\]^_**.

Remarks This command causes PMAC to send the specified control character over one of the communications ports. These can be used for printer and terminal control codes, or for special communications to a host computer

Control characters have ASCII byte values of 0 to 31 ($1F). The specified **{letter}** character determines which control character is sent when the statement is executed. The byte value of the control character sent is 64 ($40) less than the byte value of **{letter}**. The letters that can be used and their corresponding control characters are:

{letter}	Letter Value	Control Character	Value
@	64	NULL	0
A	65	<CTRL-A>	1
B	66	<CTRL-B>	2
C	67	<CTRL-C>	3
...			
X	88	<CTRL-X>	24
Y	89	<CTRL-Y>	25
Z	90	<CTRL-Z>	26
[	91	ESC	27
\	92		28
]	93		29
^	94		30
_	95		31

Note: Do not put the up-arrow character and the letter in quotes (do not use **SEND"^A"**) or PMAC will attempt to send the two non-control characters **^** and **A** for this example, instead of the control character.

SEND transmits over the active communications response port, whether serial, parallel host port (PC-Bus or STD-Bus), or VME-Bus port.

SENDS always transmits over the serial port regardless of which port is the current active response port.

SENDP always transmits over the parallel host port (PC-Bus or STD-Bus), regardless of which port is the current active response port.

There is no **SENDV** command for the VME bus exclusively. The **SEND** command must be used with the VME port as the active response port.

When PMAC powers up or resets, the active response port is the serial port. When any command is received over a bus port, the active response port becomes the bus port. PMAC must then receive a **<CONTROL-Z>** command to cause the response port to revert back to the serial port.

It is possible, particularly in PLC programs, to order the sending of messages faster than the port can handle them. Usually this will happen if the same **SEND** command is executed through every scan in the PLC. For this reason, it is good practice to have at least one of the conditions that causes the **SEND** command to execute to be set false immediately to prevent execution of this **SEND** command on subsequent scans of the PLC.

See Also On-line command **<CTRL-Z>**
Program commands **SEND"{message}"**, **COMMAND"{command}"**, **COMMAND^{letter}**

SETPHASE

Function Set motor commutation phase-position register(s)

Type Motion program, PLC program

Syntax
```
SETPHASE{constant}[,{constant}...]
SETPHASE{constant}..{constant}
[,{constant}..{constant}...]
```
where:

- **{constant}** is an integer from one to eight representing a motor number.

Remarks This command causes PMAC to force the commutation phase-position register for the specified motor or motor's to the value of the Ix75 phase-position offset parameter.

The main use of this command is to correct the phase position value at a known position (usually the motor home position) after an approximate phasing search or phasing read (e.g. from Hall commutation sensors). The approximate referencing is sufficient to move to a known position, but not necessarily to get peak performance from the motor.

This command forces a value into an "actual" (not "commanded") position register, so it is important that the actual position value be known with precision, either due to small following error or quick reaction to an actual-position trigger.

In a motion program, this command executes immediately at program calculation (lookahead) time, so for proper use in a motion program, it must be preceded by a **DWELL** command and/or an in-position loop. If a motor specified in the statement is not assigned to the coordinate system executing the program, the action will not be executed for that motor (but no error will be reported).

Example Motion program:
```
HOME1..3
WHILE(M180=0) WAIT          ; Wait for all in-position
SETPHASE1..3                ; Force phase values in
```
PLC program:
```
CMD"#4$"                    ; Rough phase search/read
WHILE(M440=0)               ; Wait for motor in-position
ENDWHILE
```

```
CMD"#4HM"                ; Homing searach move
WHILE(M440=0)            ; Wait for motor in-position
ENDWHILE
SETPHASE4                ; Force phase value in
```

See Also Power-On Phasing Search (Setting Up PMAC Commutation)
I-variables Ix75, Ix81
On-Line Command **SETPHASE**

SPLINE1

Function Put program in uniform cubic spline motion mode

Type Motion program (PROG and ROT)

Syntax **SPLINE1**

Remarks This modal command puts the program in cubic spline mode. In **SPLINE1** mode, each programmed move takes **TA** time (Ix87 is default) – there is no feedrate specification allowed. Each move on each axis is computed as a cubic position trajectory in which the intermediate positions are relaxed somewhat so there are no velocity or acceleration discontinuities in blending the moves together.

Before the first move in any series of consecutive moves, a starting move of **TA** time is added to blend smoothly from a stop. After the last move in any series of consecutive moves, an ending move of **TA** time is added to blend smoothly to a stop. If the **TA** time is changed in the middle of a series of moves, there will be a stop generated, with an extra $\mathbf{TA}_1$ move and an extra $\mathbf{TA}_2$ move added.

This command will take the program out of any of the other move modes (**LINEAR**, **CIRCLE**, **PVT**, **RAPID**). The program will stay in this mode until another move mode command is executed.

Example

```
RAPID X10 Y10
SPLINE1 TA100
X20 Y15
X32 Y21
X43 Y26
X50 Y30
DWELL100
RAPID X0 Y0
```

See Also Cubic Spline Mode (Writing a Motion Program)
I-variable Ix87
Program commands **LINEAR**, **CIRCLE**, **RAPID**, **PVT**, **SPLINE2**, **TA**

SPLINE2

Function Put program in non-uniform cubic spline motion mode

Type Motion program (PROG and ROT)

Syntax **SPLINE2**

Remarks This modal command puts the program in non-uniform cubic spline mode. This mode is virtually identical to the **SPLINE1** uniform cubic spline mode described above, except that the TA segment time can vary in a continuous spline. This makes **SPLINE2** mode more flexible than **SPLINE1** mode, but it takes slightly more computation time.

Example

```
RAPID X10 Y10
SPLINE2
X20 Y15 TA100
```

```
X32 Y21 TA120
X43 Y26 TA87
X50 Y30 TA62
DWELL100
RAPID X0 Y0
```

See Also Cubic Spline Mode (Writing a Motion Program)
I-variable Ix87
Program commands **LINEAR**, **CIRCLE**, **RAPID**, **PVT**, **SPLINE1**, **TA**

STOP

Function Stop program execution

Type Motion program (PROG)

Syntax **STOP**

Remarks This command suspends program execution, whether started by run or step, keeping the program counter pointing to the next line in the program, so that execution may be resumed with a **RUN** or **STEP** command.

Example

```
A10 B10
A20 B0
STOP
A0 B0
```

See Also On-line commands **<CONTROL-Q>**, **Q**, **R**, **S**
Program commands **BLOCKSTART**, **BLOCKSTOP**

T{data}

Function Tool Select Code (T-Code)

Type Motion program

Syntax **T{data}**
where

- **{data}** is a floating-point constant or expression in the range 0.000 to 999.999, specifying the program number and the line label to jump to.

Remarks PMAC interprets this statement as a **CALL 10n2.({data'}*1000)** command, where **n** is the hundreds' digit of **{data}**, and **{data'}** is the value of **{data}** without the hundred's digit (modulo 100 in mathematical terms). That is, this statement causes a jump (with return) to motion program 10n2, and the specified line label. (Programs 10n2 are usually used to implement the machine codes as the system designer sees fit.) The value of **{data'}** can be from 0.0 to 99.999, corresponding to line labels **N0** to **N99999**.

If the specified program and/or line label do not exist, the **T** command is ignored, and the program continues as if it were not there. No error is generated.

This structure permits the implementation of customizable T-code routines for machine-tool style applications by the writing of subroutines in motion programs 10n2. Arguments can be passed to these subroutines by following the T-code with one or more sets of **{letter}{data}**, as in **CALL** and **READ** statements.

Most users will have T-codes only in the range 0-99, which permits the use of PROG 1002 only, and allows **{data'}** to equal **{data}** for direct specification of the line label.

Example
T01 jumps to **N1000** of PROG 1002
T12 jumps to **N12000** of PROG 1002
T115 jumps to **N15000** of PROG 1012

See Also Program commands **CALL{data}, D{data}, M{data}, T{data}, RETURN**

TA{data}

Function Set Acceleration Time

Type Motion program (PROG and ROT)

Syntax **TA{data}**
where:

- **{data}** is a constant or expression representing the acceleration time in milliseconds

Remarks This statement specifies the commanded acceleration time between blended moves (**LINEAR** and **CIRCLE** mode), and from and to a stop for these moves. In **PVT** and **SPLINE1** mode moves, which are generally continually accelerating and decelerating, it specifies the actual move segment time. The units are milliseconds. PMAC will round the specified value to the nearest integer number of milliseconds when executing this command (no rounding is done in storing the value in the buffer).

Note:

Make sure the specified acceleration time (TA or 2*TS) is greater than zero, even if you are planning to rely on the maximum acceleration rate parameters (Ix17). A specified acceleration time of zero will cause a divide-by-zero error. The minimum specified time should be **TA1 TS0**.

If the specified S-curve time (from TS, or Ix88) is greater than half the TA time, the time used for the acceleration for blended moves will be twice the specified S-curve time.

The acceleration time is also the minimum time for a blended move. If the distance on a feedrate-specified (**F**) move is so short that the calculated move time is less than the acceleration time, or the time of a time-specified (**TM**) move is less than the acceleration time, the move will be done in the acceleration time instead. This will slow down the move. If **TA** controls the move time, it must be greater than the I13 time and the I8 period.

Note:

The acceleration time will be extended automatically when any motor in the coordinate system is asked to exceed its maximum acceleration rate (Ix17) for a programmed **LINEAR** mode move with I13=0 (no move segmentation).

A move executed in a program before any **TA** statement will use the default acceleration time specified by coordinate system I-variable Ix87.

In executing the TA command, PMAC rounds the specified value to the nearest integer number of milliseconds (there is no rounding done when storing the command in the buffer).

Example

```
TA100
TA(P20)
TA(45.3+SQRT(Q10))
```

See Also Linear, Circular Blended Moves, Cubic Spline Moves, PVT Moves (Writing a Motion Program)
I-variables Ix17, Ix87, Ix88
Program commands **TS**, **TM**, **PVT**.

TINIT

Function Initialize selected transformation matrix

Type Motion program (PROG and ROT)

Syntax **TINIT**

Remarks This command initializes the currently selected (with **TSEL**) transformation matrix for the coordinate system by setting it to the identity matrix. This makes the rotation angle 0, the scaling 1, and the displacement 0, so the XYZ points for the coordinate system are as the axis definition statements created them. PMAC will still perform the matrix calculations, even though they have no effect. **TSEL0** should be used to stop the matrix calculations

The matrix can subsequently be changed with the **ADIS**, **IDIS**, **AROT**, and **IROT** commands.

Example

```
TSEL 4                ; Select transformation matrix 4
TINIT                 ; Initialize it to the identity matrix
IROT 71               ; Do incremental rotation/scaling with Q71-Q79
```

See Also Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **TSEL**, **ADIS**, **IDIS**, **AROT**, **IROT**

TM{data}

Function Set Move Time

Type Motion program

Syntax **TM{data}**
where:

- **{data}** is a floating-point constant or expression representing the move time in milliseconds. The maximum effective **TM** value is 2^{23} msec. The minimum effective **TM** value is 1 msec.

Remarks This command establishes the time to be taken by subsequent **LINEAR** or **CIRCLE** mode (blended) motions. It overrides any previous **TM** or **F** statement, and is overridden by any subsequent **TM** or **F** statement. It is irrelevant in **RAPID**, **SPLINE**, and **PVT** move modes, but the latest value will stay active through those modes for the next return to blended moves.

The acceleration time is the minimum time for a blended move. If the specified move time is shorter than the acceleration time, the move will be done in the acceleration time instead. This will slow down the move. If **TM** controls the move time it must be greater than the I13 time and the I8 period.

> ***Note:***
>
> For **LINEAR** mode moves with I13=0 (no move segmentation), if the commanded velocity (distance/TM) of any motor in the move exceeds its maximum limit (Ix16), all motors in the coordinate system will be slowed down in proportion so that no motor exceeds its limit.

Example

```
TM30
TM47.635
TM(P1/3)
```

See Also Linear and Circular Blended Moves (Writing a Motion Program)
I-variable Ix16
Program commands **F, TA, TS, LINEAR, CIRCLE**

TS{data}

Function Set S-Curve Acceleration Time

Type Motion program (PROG and ROT)

Syntax **TS{data}**
where:

- **{data}** is a positive constant or expression representing the S-curve time in milliseconds.

Remarks This command specifies the time, at both the beginning and end of the total acceleration time, in **LINEAR** and **CIRCLE** mode blended moves that is spent in S-curve acceleration.

If **TS** is zero, the acceleration is constant throughout the **TA** time and the velocity profile is trapezoidal. If **TS** is greater than zero, the acceleration will start at zero and linearly increase through **TS** time, then stay constant (for time **TC**) until **TA-TS** time, and linearly decrease to zero at **TA** time (that is, **TA=2TS+TC**). If **TS** is equal to **TA/2**, the entire acceleration will be spent in S-curve form (**TS** values greater than **TA/2** override the **TA** value. Total acceleration time will be **2TS**.

> ***Note:***
>
> For **LINEAR** mode moves with PMAC not in segmentation mode (I13=0), if the rate of acceleration for any motor in the coordinate system exceeds that motor's maximum as specified by Ix17, the acceleration time for all motors is increased so that no motor exceeds its maximum acceleration rate.

TS does not affect **RAPID**, **PVT**, or **SPLINE** mode moves, but it stays valid for the next return to blended moves.

> ***Note:***
>
> Make sure the specified acceleration time (TA or 2*TS) is greater than zero, even if planning to rely on the maximum acceleration rate parameters (Ix17). A specified acceleration time of zero will cause a divide-by-zero error. The minimum specified time should be **TA1 TS0**.

In executing the **TS** command, PMAC rounds the specified value to the nearest integer number of milliseconds (there is no rounding done when storing the command in the buffer).

A blended move executed in a program before any **TS** statement will use the default S-curve time specified by coordinate system I-variable Ix88.

Example

```
TS20
TS(Q17)
```

TS(39.32+P43)

See Also Linear and Circular Blended Moves (Writing a Motion Program)
I-variables I13, Ix17, Ix21, Ix87, Ix88
Program commands **TA**, **TM**, **F**, **LINEAR**, **CIRCLE**

TSELECT{constant}

Function Select active transformation matrix for X, Y, and Z axes

Type Motion program (PROG and ROT)

Syntax **TSELECT{constant}**
where:

- **{constant}** is an integer representing the number of the matrix to be used.

Remarks This command selects the specified matrix for use as the active transformation matrix for the X, Y, and Z axes of the coordinate system running the motion program. This matrix can then be modified using the **TINIT**, **ADIS**, **AROT**, **IDIS**, and **IROT** commands to perform translations, rotations, and scaling of the three axes. This matrix will be used until another one is selected.

This matrix must already have been created with the on-line **DEFINE TBUF** command. That command specifies the number of matrices to create, and it must have specified a number at least as high as the number used in **TSEL** (you cannot select a matrix that has not been created).

TSEL0 deselects all transformation matrices, saving calculation time.

Example

```
DEFINE TBUF 5                    ; Create five transformation matrices
OPEN PROG 10 CLEAR
...
TSEL 3                           ; Select transformation matrix 3 (of 5)
TINIT                            ; Make matrix 3 the identity matrix
```

See Also Axis Matrix Transformations (Writing a Motion Program)
On-line command **DEFINE TBUF**
Program commands **AROT**, **IROT**, **ADIS**, **IDIS**, **TINIT**

U{data}

Function U-Axis Move

Type Motion program

Syntax **U{data}**
where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the U-axis.

Remarks This command causes a move of the U-axis. (See **{axis}{data}** description, above.)

Example

```
U10
U(P17+2.345)
X20 U20

U(COS(Q10)) V(SIN(Q10))
```

See Also Program commands **{axis}{data}**, **A**, **B**, **C**, **V**, **W**, **X**, **Y**, **Z**, **CALL**, **READ**

V{data}

Function V-Axis Move

Type Motion program (PROG and ROT)

Syntax **V{data**
where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the V-axis.

Remarks This command causes a move of the V-axis. (See **{axis}{data}** description, above.)

Example
```
V20
U56.5 V(P320)
Y10 V10
V(SQRT(Q20*Q20+Q21*Q21))
```

See Also Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **W**, **X**, **Y**, **Z**, **CALL**, **READ**

W{data}

Function W-Axis Move

Type Motion program

Syntax **W{data}**
where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the W-axis.

Remarks This command causes a move of the W-axis. (See **{axis}{data}** description, above.)

Example
```
W5
W(P10+33.5)
Z10 W10
W(ABS(Q22*Q22))
```

See Also Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **V**, **X**, **Y**, **Z**, **CALL**, **READ**

WAIT

Function Suspend program execution

Type Motion program (PROG and ROT)

Syntax **WAIT**

Remarks This command may be used on the same line as a **WHILE** condition to hold up execution of the program until the condition goes false. When the condition goes false, program execution resumes on the next line. Use of the **WAIT** statement allows indefinite pauses without the need for repeated use of a servo command (e.g., **DWELL** or **DELAY**) to eat up the time. However, it is impossible to predict how long the pause will be.

WAIT permits a faster resumption of the program upon the **WHILE** condition going false. Also, the program timer is halted when **WAIT**ing, which allows the "In-position" bit to go true (which can be used to trigger an action, or the next move).

Since PMAC executes a **WHILE ({condition}) WAIT** statement every Real Time Interrupt until the condition goes false, it is essentially the same as a PLC0. This could use excessive processor time and in severe cases trip the watchdog timer on PMACs that simultaneously run several motion programs that use **WAIT** statements and or large PLC0 programs.

For example, if the condition only needs to be checked every 20 msec and not every Real Time Interrupt, you should consider using a **DWELL** command to regulate the execution time of the **WHILE** loop.

```
WHILE ({condition})
      DWELL20
ENDW
```

Example

```
WHILE (M11=0) WAIT          ; Pause here until Machine Input 1 set
WHILE (M187=0) WAIT         ; Pause here until all axes in-position
M1=1                        ; Turn on Output 1 to activate punch
```

See Also

I-variable Ix28
Program commands **DWELL**, **DELAY**, **STOP**

WHILE({condition})

Function

Conditional looping

Type

Motion program (PROG only); PLC program

Syntax

```
WHILE ({condition})
WHILE ({condition}) {action}
```

where:

- **{condition}** consists of one or more sets of **{expression} {comparator} {expression}**, joined by logical operators **AND** or **OR**.
- **{action}** is a program command.

Remarks

This statement allows repeated execution of a statement or series of statements as long as the condition is true. It is PMAC's only looping construct. It can take two forms:

1. (Valid in motion program only) With a statement following on the same line, it will repeatedly execute that statement as long as the condition is true. No **ENDWHILE** is used to terminate the loop.

   ```
   WHILE ({condition}) {action}
   ```

2. (Valid in motion and PLC programs) With no statement following on the same line, it will execute statements on subsequent lines down to the next **ENDWHILE** statement.

   ```
   WHILE ({condition})
         {statement}
         [{statement}
         ...]
   ENDWHILE
   ```

If a **WHILE** loop in a motion program has no move, **DWELL**, or **DELAY** inside, PMAC will attempt to execute the loop twice (while true) each real-time interrupt cycle (stopped from more loops only by the "double-jump-back" rule), much like a PLC0. This can starve the background tasks for time, possibly even tripping the watchdog timer. PMAC will not attempt to blend moves through such an "empty" **WHILE** loop if it finds the loop condition true twice or more.

In PLC programs, extended compound **WHILE** conditions can be formed on multiple program lines through use of **AND** and **OR** commands on the program lines immediately following the **WHILE** command itself (this structure is not available in motion programs). Conditions in each program line can be either simple or compound. **AND** and **OR** operations within a program line take precedence over **AND** and **OR** operations between lines.

Example

```
WHILE (P20=0)
   ...
ENDWHILE
WHILE (Q10<5 AND Q11>1)
   ...
ENDWHILE
WHILE (M11=0) WAIT                    ; sit until input goes true
INC
WHILE (M11=0 OR M12=0) X100           ; increment until  2 inputs true
```

To do the equivalent of a For/Next loop:

```
P1=0                                  ; Initialize loop counter
WHILE (P1<10)                         ; Loop until counter exceeds limit
    X1000                             ; Perform action to be repeated
    P1=P1+1                           ; Increment loop counter
ENDWHILE                              ; Loop back
```

To do a timed wait in a PLC program, use the servo cycle counter as timer

```
P90=16777216                          ; Counter rollover value (2^24)
P91=M0                                ; Store starting value of M0 (X:$0) counter
P92=0                                 ; Time elapsed so far
WHILE (P92<P93)                       ; Loop until past specified time
   P92=(M0-P91)%P90                   ; Calculate time elapsed
                                      ; Modulo (%) operation to handle rollover
ENDWHILE                              ; Loop back
```

To do extended compound conditions in a PLC program

```
WHILE (M11=1 AND M12=1)
OR (M13=1 AND M14=1)
AND (P1>0)
   ...
ENDWHILE
```

See Also

Program Logic (Writing a Motion Program, Writing a PLC Program)
How PMAC Executes a Motion Program (Writing a Motion Program)
Program commands **AND**, **OR**, **IF**, **ELSE**, **ENDIF**, **ENDWHILE**

X{data}

Function X-Axis Move

Type Motion program

Syntax **X{data}**

where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the X-axis.

Remarks This command causes a move of the X-axis. (See **{axis}{data}** description above.)

Example

```
X10
X15 Y20
X(P1) Y30
X(Q10*COS(Q1)) Y(Q10*SIN(Q1))
X3.76 Z2.92 I0.075 K3.42
```

See Also Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **V**, **W**, **Y**, **Z**, **CALL**, **READ**

Y{data}

Function Y-Axis Move

Type Motion program

Syntax **Y{data}**
where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the Y-axis.

Remarks This command causes a move of the Y-axis. (See **{axis}{data}** description above.)

Example
```
Y50
Y(P100)
X35 Y75
Y-0.221 Z3.475
Y(ABS(P3+P4)) A(INT(P3-P4))
```

See Also Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **V**, **W**, **X**, **Z**, **CALL**, **READ**

Z{data}

Function Z-Axis Move

Type Motion program

Syntax **Z{data}**
where:

- **{data}** is a floating point constant or expression representing the position or distance in user units for the W-axis.

Remarks This command causes a move of the Z-axis. (See **{axis}{data}** description above.)

Example
```
Z20
Z(Q25)
X10 Y20 Z30
Z23.4 R10.5
Z(P301+2*P302/P303)
```

See Also Program commands **{axis}{data}**, **A**, **B**, **C**, **U**, **V**, **W**, **X**, **Y**, **CALL**, **READ**

PMAC MATHEMATICAL FEATURES

Mathematical Operators

+

Function Addition

Remarks The + sign implements the addition of the numerical values preceding and following it.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

The + sign may not be used as a "unary" operator to emphasize that the positive value of the following variable or constant is to be used (e.g. **P1=+P2**). This syntax will be rejected with an error.

-

Function Subtraction

Remarks The - sign implements the subtraction of the numerical value following it from the numerical value preceding it. If there is no numerical value immediately preceding it, it causes the negation of the numerical value following it (e.g. **P1=-P2**).

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

*

Function Multiplication

Remarks The * sign implements the multiplication of the numerical values preceding and following it.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

/

Function Division

Remarks The / sign implements the division of the numerical value preceding it by the numerical value following it. Unless the division is executed in a compiled PLC on a line with only L-variables and integers, the division operation is always a floating-point calculation (even if integer values are used). The quotient is computed as a floating-point value and used as such in subsequent calculations in the same expression; if it is then stored to an integer, it is rounded at the time of storage.

If the division operation is performed as an integer operation in a compiled PLC (only L-variables and integer constants on the line), the quotient is computed as an integer (rounded to the nearest integer value) and used as such in subsequent calculations in the same expression.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

If the divisor is equal to 0, the result will saturate at +/-2^{47} (+/-2^{23} for an integer division in a compiled PLC). No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible division-by-zero errors.

Example

Command	Result
P1=10*2/3	6.666666667
P1=10*(2/3)	6.666666667
M1=10*2/3	7
M1=10*(2/3)	7
L1=10*2/3	7
L1=10*(2/3)	10
L1 and M1 are integer variables; P1 is a floating-point variable	

%

Function Modulo (remainder)

Remarks The % sign causes the calculation of the remainder due to the division of the numerical value preceding it by the numerical value following it. Unless the division is executed in a compiled PLC on a line with only L-variables and integers, the division operation is always a floating-point calculation (even if integer values are used). The quotient is computed as a floating-point value, then truncated to the next lowest (i.e., toward -∞) integer so the remainder can be computed.

If the divisor "n" is a positive value, the modulo result is in the range {0 ≤ Result < n}. If the divisor "n" is a negative value, the modulo result is in the range {–n ≤ Result < n}.

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

If the divisor is equal to 0, the division will saturate and the modulo result will be 0. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible division-by-zero errors.

Example

Operation	Result
11%4	3
-11%4	1
11%-4	3
-11%-4	-3
3%2.5	0.5
-3%2.5	2
3%-2.5	-2
-3%-2.5	2

&

Function Bit-by-bit "and"

Remarks The & sign implements the bit-by-bit logical “and” of the numerical value preceding it and the numerical value following it. A given bit of the result is equal to 1 if the matching bits of both operands are equal to 1. The operation is done both on integer bits and fractional bits (if any).

Multiplication, division, modulo (remainder), and bit-by-bit “and” operations have higher priority than addition, subtraction, bit-by-bit “or”, and bit-by-bit “exclusive-or” operations. Operations of the same priority are implemented from left to right.

This bit-by-bit “and” operator that logically combines the bits of numerical values is not to be confused with the **AND** command, which logically combines conditions.

Example

Operation	Result
3&1	1
3&2	2
3&3	3
3&4	0
3&-3	1
0.875&1.75	0.75
0.875&-1.75	0.25

|

Function Bit-by-bit "or"

Remarks The | sign implements the bit-by-bit logical “or” of the numerical value preceding it and the numerical value following it. A given bit of the result is equal to 1 if the matching bit of either operand is equal to 1. The operation is done both on integer bits and fractional bits (if any).

Multiplication, division, modulo (remainder), and bit-by-bit “and” operations have higher priority than addition, subtraction, bit-by-bit “or”, and bit-by-bit “exclusive-or” operations. Operations of the same priority are implemented from left to right.

This bit-by-bit “or” operator that logically combines the bits of numerical values is not to be confused with the **OR** command, which logically combines conditions.

Example

Operation	Result
4|3	7
3|2	3
3|3	3
$F0|$4	$F4
3|-3	-1
0.5|0.375	0.875
0.875|-1.75	-0.375

^

Function "Bit-by-bit "exclusive or"

Remarks The ^ sign implements the bit-by-bit logical "exclusive or" (xor) of the numerical value preceding it and the numerical value following it. A given bit of the result is equal to 1 if the matching bits of the two operands are different from each other. The operation is done both on integer bits and fractional bits (if any).

Multiplication, division, modulo (remainder), and bit-by-bit "and" operations have higher priority than addition, subtraction, bit-by-bit "or", and bit-by-bit "exclusive-or" operations. Operations of the same priority are implemented from left to right.

Example

Operation	Result
2^1	3
2^2	0
5^7	2
$AA^$55	$FF
3^-3	-2
0.5^0.875	0.375

Mathematical Functions

ABS

Function Absolute value

Syntax **ABS({expression})**

Domain All real numbers

Domain units User-determined

Range Non-negative real numbers

Range units User-determined

Remarks **ABS** implements the absolute-value, or magnitude function of the mathematical expression contained inside the following parentheses.

Example

```
P8=ABS(P7)                    ; Computes magnitude of P7
IF(Q200!=0)                   ; Divide by 0 check
  Q240=ABS(Q200)/Q200         ; Computes sign (-1 or 1) of Q200
ELSE
  Q240=0                      ; Sign value is 0
ENDIF
```

ACOS

Function Trigonometric arc-cosine

Syntax **ACOS({expression})**

Domain -1.0 to +1.0

Domain units none

Range 0 to Pi radians (0 to 180 degrees)

Range units Radians/degrees

Remarks **ACOS** implements the inverse cosine, or arc-cosine function of the mathematical expression contained inside the following parentheses.

This function returns values in degrees if I15 is set to the default value of 0; it returns values in radians if I15 is set to 1.

If the argument inside the parentheses is outside of the legal domain of –1.0 to +1.0, an arbitrary value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Example

```
P50=ACOS(P48/P49)          ; Computes angle whose cos is P48/P49
C(ACOS(Q70/10))            ; Move C axis to specified angle
```

ASIN

Function Trigonometric arc-sine

Syntax **ASIN({expression})**

Domain -1.0 to +1.0

Domain units none

Range 0 to Pi radians (0 to 180 degrees)

Range units Radians/degrees

Remarks **ASIN** implements the inverse sine, or arc-sine function of the mathematical expression contained inside the following parentheses.

This function returns values in degrees if I15 is set to the default value of 0; it returns values in radians if I15 is set to 1.

If the argument inside the parentheses is outside of the legal domain of –1.0 to +1.0, an arbitrary value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Example

```
P50=ASIN(P48/P49)          ; Computes angle whose sin is P48/P49
C(ASIN(Q70/10))            ; Move C axis to specified angle
```

ATAN

Function Trigonometric arc-tangent

Syntax **ATAN({expression})**

Domain All real numbers

Domain units none

Range -Pi/2 to +Pi/2 radians (-90 to +90 degrees)

Range units Radians/degrees

Remarks **ATAN** implements the standard inverse tangent, or arc-tangent function of the mathematical expression contained inside the following parentheses. This standard arc-tangent function returns values only in the +/-90-degree range; if a full +/-180-degree range is desired, the **ATAN2** function should be used instead.

This function returns values in degrees if I15 is set to the default value of 0; it returns values in radians if I15 is set to 1.

Example

```
P50=ATAN(P48/P49)          ; Computes angle whose tan is P48/P49
C(ATAN(Q70/10))            ; Move C axis to specified angle
```

ATAN2

Function Two-argument trigonometric arc-tangent

Syntax **ATAN2({expression})**

Domain All real numbers in both arguments

Domain units none

Range -Pi to +Pi radians (-180 to +180 degrees)

Range units Radians/degrees

Remarks **ATAN2** implements the expanded (two-argument) inverse tangent, or arc-tangent function of the mathematical expression contained inside the following parentheses, and the value of variable Q0 for the coordinate system used. (If this function is used inside a PLC program, make sure the desired coordinate system has been selected with the **ADDRESS** command.)

This expanded arc-tangent function returns values in the full +/-180-degree range; if only the +/-90-degree range is desired, the standard **ATAN** function should be used instead. The **ATAN2** function makes use of the signs of both arguments, as well as the ratio of their magnitudes, to extend the range to a full 360 degrees. The value in the parentheses following **ATAN2** is the "sine" argument; the value in Q0 is the "cosine" argument.

This function returns values in degrees if I15 is set to the default value of 0; it returns values in radians if I15 is set to 1.

If both arguments for the **ATAN2** function are equal to exactly 0.0, an internal division-by-zero error will result, and an arbitrary value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for these possible domain errors.

Example

```
Q30=-0.707                 ; "Cosine" argument
Q31=-0.707                 ; "Sine" argument
Q32=ATAN(Q31/Q30)          ; Single-argument arctangent
Q32                        ; Query resulting value
45                         ; Returns value in +/-90 range
Q0=Q30                     ; Prepare "cosine" for ATAN2
Q33=ATAN2(Q31)             ; Two-argument arctangent
Q33                        ; Query resulting value
-135                       ; Note different result

Q0=M163-M161               ; X target – X present position
Q1=M263-M261               ; Y target – Y present position
IF (ABS(Q0)>0.001 OR ABS(Q1)>0.001)  ; Div by 0 check
  Q2=ATAN2(Q1)             ; Calculate directed angle
ENDIF
```

COS

Function Trigonometric cosine

Syntax **COS({expression})**

Domain All real numbers

Domain units Radians/degrees

Range -1.0 to +1.0

Range units none

Remarks **COS** implements the trigonometric cosine function of the mathematical expression contained inside the following parentheses.

This function interprets its argument in degrees if I15 is set to the default value of 0; it interprets its argument in radians if I15 is set to 1.

Example

```
P60=COS(30)              ; Computes cosine of 30
X(Q80*COS(Q81))          ; Move X axis to calculated value
```

EXP

Function Exponentiation (e^x)

Syntax **EXP({expression})**

Domain All real numbers

Domain units User-determined

Range Positive real numbers

Range units User-determined

Remarks **EXP** implements the standard exponentiation function of the mathematical expression contained inside the following parentheses, raising "e" to the power of this expression.

To implement the y^x function, use $e^{x\ ln(y)}$ instead.

Example

```
P20=EXP(P19)             ; Raises e to the power of P19
P3=EXP(P2*LN(P1))        ; Raises P1 to the power of P2
```

INT

Function Truncation to integer

Syntax **INT({expression})**

Domain All real numbers

Domain units User-determined

Range All integers

Range units User-determined

Remarks **INT** implements the truncation to integer function of the mathematical expression contained inside the following parentheses. The truncation is always done in the negative direction.

Note that while the result is an integer number, it is still represented as a floating-point value.

Example

```
P50=2.5                 ;
P51=INT(P50)            ; Take INT of positive value
P51                     ; Query resulting value
2                       ; Next lower integer value
P52=INT(-P50)           ; Take INT of negative value
P52                     ; Query resulting value
-3                      ; Next lower integer value
```

LN

Function Natural logarithm

Syntax **EXP({expression})**

Domain Positive real numbers

Domain units User-determined

Range All real numbers

Range units User-determined

Remarks **LN** implements the natural logarithm (logarithm base "e") function of the mathematical expression contained inside the following parentheses.

To implement the logarithm using another base, divide the natural logarithm of the value by the natural logarithm of the base ($\log_y x = \ln x / \ln y$). The natural logarithm of 10 is equal to 2.302585.

If the argument inside the parentheses is outside of the legal domain of positive numbers, a 0 value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Example

```
P19=LN(P20)             ; Takes the natural log of P20
P6=LN(P5)/LN(10)        ; Takes the log base 10 of P5
```

SIN

Function Trigonometric sine

Syntax **SIN({expression})**

Domain All real numbers

Domain units Radians/degrees

Range -1.0 to +1.0

Range units none

Remarks **SIN** implements the trigonometric sine function of the mathematical expression contained inside the following parentheses.

This function interprets its argument in degrees if I15 is set to the default value of 0; it interprets its argument in radians if I15 is set to 1.

Example

```
P60=SIN(30)             ; Computes cosine of 30
Y(Q80*SIN(Q81))         ; Move Y axis to calculated value
```

SQRT

Function Square root

Syntax **SQRT({expression})**

Domain Non-negative real numbers

Domain units User-determined

Range Non-negative real numbers

Range units User-determined

Remarks **SQRT** implements the positive square-root function of the mathematical expression contained inside the following parentheses.

If the argument inside the parentheses is outside of the legal domain of non-negative numbers, an arbitrary value will be returned. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Example

```
P19=SQRT(P20)                    ; Takes the square root of P20
P50=SQRT(P8*P8+P9*P9)            ; Pythagorean theorem calculation
```

TAN

Function Trigonometric tangent

Syntax **TAN({expression})**

Domain All real numbers except +/-(2N-1)*90 degrees

Domain units Radians/degrees

Range All real numbers

Range units none

Remarks **TAN** implements the trigonometric tangent function of the mathematical expression contained inside the following parentheses.

This function interprets its argument in degrees if I15 is set to the default value of 0; it interprets its argument in radians if I15 is set to 1.

If the argument inside the parentheses approaches +/-(2N-1)*90 degrees (+/-90, 270, 450, etc.), the TAN function will "blow up" and a very large value will be returned. If the argument inside the parentheses is exactly equal to one of these quantities, an internal division-by-zero error will occur and the resulting value will saturate at $+/-2^{47}$. No error will be reported, and the program will not stop. It is the programmer's responsibility to check for possible domain errors.

Example

```
P60=TAN(30)                      ; Computes cosine of 30
Y(Q80*TAN(Q81))                  ; Move Y axis to calculated value
```

SAVED SETUP REGISTERS NOT REPRESENTED BY I-VARIABLES

PMAC and PMAC2 controllers each have several setup registers that operate like I-variables, but are not represented by I-variables. The values of these setup registers are stored in non-volatile memory with the **SAVE** command, and they are restored to the active registers on a power-up/reset.

These setup registers fall in several categories: the PMAC2 analog "de-multiplexing" data table, the encoder conversion table setup registers, the VME and DPRAM addressing setup registers, and some PMAC2 Servo IC setup bits and registers. Each is detailed below.

Analog Data Table Setup Registers

PMAC2 controllers (and PMAC(1)-PCI controllers) are available with optional on-board analog-to-digital converters. Option 12 provides 8 12-bit ADCs; Option 12A provides another set of 8 12-bit ADCs. Only one of each set of 8 ADCs can be read at any time, so PMAC firmware provides a de-multiplexing scheme that automatically breaks out each channel into a separate register, where it can be read directly and transparently by software tasks, whether as servo-loop input or program use.

Note

In V1.17C and newer firmware, these 8 registers can be accessed as I70-I77.

X:$0708 – Y:$070F Analog Table Setup Lines

Range: $000000 - $FFFFFF

Units: none

Defaults: $0

PMAC2 firmware automatically selects and reads the channels of Option 12 and 12A A/D converters in a round-robin fashion. This function is controlled by a data table in registers $0708 to $070F which operates much like the encoder conversion table. The eight X registers contain the channel-select information, and the eight Y registers contain the A/D results. Each X and Y word is split into two 12-bit halves, where the lower 12 bits work with the first A/D converter set (Option 12), and the higher 12 bits work with the second A/D converter set (Option 12A).

The data table looks like this:

PMAC2 Address	X Word Upper 12 Bits	X Word Lower 12 Bits	Y Word Upper 12 Bits	Y Word Lower 12 Bits
$0708	CONFIG_W2	CONFIG_W1	DATA_W2	DATA_W1
$0709	CONFIG_W2	CONFIG_W1	DATA_W2	DATA_W1
$070A	CONFIG_W2	CONFIG_W1	DATA_W2	DATA_W1
$070B	CONFIG_W2	CONFIG_W1	DATA_W2	DATA_W1
$070C	CONFIG_W2	CONFIG_W1	DATA_W2	DATA_W1
$070D	CONFIG_W2	CONFIG_W1	DATA_W2	DATA_W1
$070E	CONFIG_W2	CONFIG_W1	DATA_W2	DATA_W1
$070F	CONFIG_W2	CONFIG_W1	DATA_W2	DATA_W1

where:

CONFIG_W2 is the selection word for the second A/D converter set (Option 12A)
CONFIG_W1 is the selection word for the first A/D converter set (Option 12)
DATA_W2 is the matching A/D data from the second A/D converter set (Option 12A)
DATA_W1 is the matching A/D data from the first A/D converter set (Option 12)

A value of 0-7 in CONFIG_W1 tells PMAC2 to read channel ANAI00-07, respectively, as a 0 to+5V input, resulting in an unsigned value.

A value of 8-15 in CONFIG_W1 tells PMAC2 to read ANAI00-07, respectively, as a -2.5 to +2.5V input, resulting in a signed value.

A value of 0-7 in CONFIG_W2 tells PMAC2 to read channel ANAI08-15, respectively, as a 0 to+5V input, resulting in an unsigned value.

A value of 8-15 in CONFIG_W1 tells PMAC2 to read ANAI08-15, respectively, as a -2.5 to +2.5V input, resulting in a signed value.

Each phase update (9 kHz default), PMAC2 increments through one line of the table. It copies the ADC reading(s) selected in the previous cycle into RAM, then writes the next configuration words to the ADC(s). Typically, this will be used to cycle through all 8 ADCs or pairs of ADCs. To cycle through all 8 pairs of ADCs in unsigned mode, the table should look like this:

PMAC Address	X Word Upper 12 Bits	X Word Lower 12 Bits	Y Word Upper 12 Bits	Y Word Lower 12 Bits
$0708	0	0	ANAI08	ANAI00
$0709	1	1	ANAI09	ANAI01
$070A	2	2	ANAI10	ANAI02
$070B	3	3	ANAI11	ANAI03
$070C	4	4	ANAI12	ANAI04
$070D	5	5	ANAI13	ANAI05
$070E	6	6	ANAI14	ANAI06
$070F	7	7	ANAI15	ANAI07

Suggested M-variable definitions for the configuration words are:

```
M990->X:$0708,0,24,U          ; 1st CONFIG_W1 and CONFIG_W2
M991->X:$0709,0,24,U          ; 2nd CONFIG_W1 and CONFIG_W2
M992->X:$070A,0,24,U          ; 3rd CONFIG_W1 and CONFIG_W2
M993->X:$070B,0,24,U          ; 4th CONFIG_W1 and CONFIG_W2
M994->X:$070C,0,24,U          ; 5th CONFIG_W1 and CONFIG_W2
M995->X:$070D,0,24,U          ; 6th CONFIG_W1 and CONFIG_W2
M996->X:$070E,0,24,U          ; 7th CONFIG_W1 and CONFIG_W2
M997->X:$070F,0,24,U          ; 8th CONFIG_W1 and CONFIG_W2
```

If you wanted to set up all ADCs for a unipolar (unsigned) conversion, the following commands could be issued

```
M990=$000000      ; Select ANAI00 and ANAI08 (if present) unipolar
M991=$001001      ; Select ANAI01 and ANAI09 (if present) unipolar
M992=$002002      ; Select ANAI02 and ANAI10 (if present) unipolar
M993=$003003      ; Select ANAI03 and ANAI11 (if present) unipolar
M994=$004004      ; Select ANAI04 and ANAI12 (if present) unipolar
M995=$005005      ; Select ANAI05 and ANAI13 (if present) unipolar
M996=$006006      ; Select ANAI06 and ANAI14 (if present) unipolar
M997=$007007      ; Select ANAI07 and ANAI15 (if present) unipolar
```

To set up the configuration words for bipolar analog inputs, the commands could look like this:

```
M990=$008008      ; Select ANAI00 and ANAI08 (if present) bipolar
M991=$009009      ; Select ANAI01 and ANAI09 (if present) bipolar
M992=$00A00A      ; Select ANAI02 and ANAI10 (if present) bipolar
M993=$00B00B      ; Select ANAI03 and ANAI08 (if present) bipolar
M994=$00C00C      ; Select ANAI04 and ANAI08 (if present) bipolar
M995=$00D00D      ; Select ANAI05 and ANAI08 (if present) bipolar
M996=$00E00E      ; Select ANAI06 and ANAI08 (if present) bipolar
M997=$00F00F      ; Select ANAI07 and ANAI08 (if present) bipolar
```

Once this setup has been made, PMAC2 will automatically cycle through the analog inputs, copying the converted digital values into RAM. These image registers can then be read as if they were the actual A/D converters. For user program use, the image registers would be accessed with M-variables. Suggested definitions for unipolar (unsigned) values are:

```
M1000->Y:$0708,0,12,U   ; ANAI00 image register; from J1 pin 1
M1001->Y:$0709,0,12,U   ; ANAI01 image register; from J1 pin 2
M1002->Y:$070A,0,12,U   ; ANAI02 image register; from J1 pin 3
M1003->Y:$070B,0,12,U   ; ANAI03 image register; from J1 pin 4
M1004->Y:$070C,0,12,U   ; ANAI04 image register; from J1 pin 5
M1005->Y:$070D,0,12,U   ; ANAI05 image register; from J1 pin 6
M1006->Y:$070E,0,12,U   ; ANAI06 image register; from J1 pin 7
M1007->Y:$070F,0,12,U   ; ANAI07 image register; from J1 pin 8
M1008->Y:$0708,12,12,U  ; ANAI08 image register; from J1 pin 9
M1009->Y:$0709,12,12,U  ; ANAI09 image register; from J1 pin 10
M1010->Y:$070A,12,12,U  ; ANAI10 image register; from J1 pin 11
M1011->Y:$070B,12,12,U  ; ANAI11 image register; from J1 pin 12
M1012->Y:$070C,12,12,U  ; ANAI12 image register; from J1 pin 13
M1013->Y:$070D,12,12,U  ; ANAI13 image register; from J1 pin 14
M1014->Y:$070E,12,12,U  ; ANAI14 image register; from J1 pin 15
M1015->Y:$070F,12,12,U  ; ANAI15 image register; from J1 pin 16
```

For bipolar (signed), just change the **U** in each definition to **S**.

Encoder Conversion Table Setup Registers: Y:$0720 – Y:$073F

The encoder conversion table (ECT), which performs pre-processing of feedback and master position data for servo-loop use, has 32 saved setup registers.

Y:$0720 – Y:$073F Conversion Table Setup Lines

Range: $000000 - $FFFFFF

Units: Modified PMAC Addresses

Defaults:

PMAC(1) Defaults

Reg.	Setting	Meaning	Reg.	Setting	Meaning
Y:$0720	$00C000	1/T Extension of Encoder 1	Y:$0725	$00C014	1/T Extension of Encoder 6
Y:$0721	$00C004	1/T Extension of Encoder 2	Y:$0726	$00C018	1/T Extension of Encoder 7
Y:$0722	$00C008	1/T Extension of Encoder 3	Y:$0727	$00C01C	1/T Extension of Encoder 8
Y:$0723	$00C00C	1/T Extension of Encoder 4	Y:$0728	$400723	Time base from 1/T of Enc 4
Y:$0724	$00C010	1/T Extension of Encoder 5	Y:$0729	$000295	Time base scale factor
Y:$072A – Y:$073F = 0					

PMAC2 Defaults

Reg.	Setting	Meaning	Reg.	Setting	Meaning
Y:$0720	$00C000	1/T Extension of Encoder 1	Y:$0725	$00C028	1/T Extension of Encoder 6
Y:$0721	$00C008	1/T Extension of Encoder 2	Y:$0726	$00C030	1/T Extension of Encoder 7
Y:$0722	$00C010	1/T Extension of Encoder 3	Y:$0727	$00C038	1/T Extension of Encoder 8
Y:$0723	$00C018	1/T Extension of Encoder 4	Y:$0728	$00C090	1/T Extension of Handwheel 1
Y:$0724	$00C020	1/T Extension of Encoder 5	Y:$0729	$00C098	1/T Extension of Handwheel 2
Y:$072A – Y:$073F = 0					

Turbo PMAC2 Ultralite Defaults

I-Var.	Setting	Meaning	I-Var.	Setting	Meaning
Y:$0720	$28C0A0	MACRO Node 0 Reg. 0 Unshifted Read	Y:$0728	$28C0B0	MACRO Node 8 Reg. 0 Unshifted Read
Y:$0721	$FFFFFF	Use all 24 bits	Y:$0729	$FFFFFF	Use all 24 bits
Y:$0722	$28C0A4	MACRO Node 1 Reg. 0 Unshifted Read	Y:$072A	$28C0B4	MACRO Node 9 Reg. 0 Unshifted Read
Y:$0723	$FFFFFF	Use all 24 bits	Y:$072B	$FFFFFF	Use all 24 bits
Y:$0724	$28C0A8	MACRO Node 4 Reg. 0 Unshifted Read	Y:$072C	$28C0B8	MACRO Node 12 Reg. 0 Unshifted Read
Y:$0725	$FFFFFF	Use all 24 bits	Y:$072D	$FFFFFF	Use all 24 bits
Y:$0726	$28C0AC	MACRO Node 5 Reg. 0 Unshifted Read	Y:$072E	$28C0BC	MACRO Node 13 Reg. 0 Unshifted Read
Y:$0727	$FFFFFF	Use all 24 bits	Y:$072F	$FFFFFF	Use all 24 bits
Y:$0730 – Y:$073F = 0					

Y:$0720 to Y:$073F form the 32 setup lines of the PMAC's Encoder Conversion Table (ECT). The main purpose of the ECT is to provide a pre-processing of feedback and master data to prepare it for use by the servo loop. It can also be used to execute certain simple calculations at the servo update frequency.

Each setup line occupies a fixed register in the PMAC's memory map. The register addresses are important, because the results of the ECT are accessed by address.

The ECT has two halves: setup and results. The setup half resides in PMAC's Y-memory, and can be accessed through these 32 setup registers. The result half resides in PMAC's X-memory. Each of the 32 setup lines has a matching result X-register at the same numerical address. If the entry consists of more than one line, the last line has the final result; any previous lines contain intermediate results.

Table Structure: The ECT consists of a series of entries, with each entry creating one processed (converted) feedback value. An entry in the ECT can have 1, 2, or 3 lines, with each line containing a 24-bit setup word in Y-memory, and a 24-bit result register in X-memory. Therefore, each entry contains 1, 2, or 3 of these 24-bit setup lines, each usually represented as a hexadecimal value with six digits. The final result is always in the X-memory register matching the *last* setup line in the entry.

The variables that commonly contain the address of the last line of the entry are Ix03 Motor x Position-Loop Feedback Address, Ix04 Motor x Velocity-Loop Feedback Address, Ix05 Motor x Master Position Address and Ix93 Coordinate System x Time-Base Address.

Entry First Line: The first line's setup register in each entry consists of a source address in the low 16 bits (bits 0 – 15, the last four hex digits), which contains the PMAC address of the raw data to be processed, a digit (the second hex digit, bits 16 - 19) that specifies how the source data is to be shifted and whether the result is to be summed with the result of the above entry, and a method value in the high 4 bits (first hex digit), which specifies how this data is to be processed. If the first line in the entry is $000000, this signifies the end of the active table, regardless of what subsequent entries in the table (higher addressed registers) contain.

Entry Additional Lines: Depending on the method, 1 or 2 additional lines may be required in the entry to provide further instructions on processing.

The following table summarizes the content of entries in the Encoder Conversion Table:

Method Digit	# of lines	Process Defined	Second digit	1st Additional Line	2nd Additional Line
$0	1	1/T Extension of Incremental Encoder	$0 = normal conversion $1 = summed with above	-	-
$1	1	ACC-28 style A/D converter (high 16 bits, no rollover)	$0 = normal, signed ADC $1 = summed, signed ADC $8 = normal, unsigned ADC $9 = summed, unsigned ADC	-	-
$2	2	Parallel Y-word data, no filtering	$0 = normal conversion $1 = summed with above $8 = unshifted conversion $9 = unshifted, summed	Bits Used Mask Word	-
$3	3	Parallel Y-word data, with filtering	$0 = normal conversion $1 = summed with above $8 = unshifted conversion $9 = unshifted, summed	Bits Used Mask Word	Max Change per Cycle
$4	2	"Time Base" scaled digital differentiation	0	Time Base Scale Factor	-
$5	2	Integrated ACC-28 style A/D converter	$0 = normal, signed ADC $8 = normal, unsigned ADC	Input Bias	-
$6	2	Parallel /X-word data, no filtering	$0 = normal conversion $1 = summed with above $8 = unshifted conversion $9 = unshifted, summed	Bits Used Mask Word	-
$7	3	Parallel X-word data, with filtering	$0 = normal conversion $1 = summed with above $8 = unshifted conversion $9 = unshifted, summed	Bits Used Mask Word	Max Change per Cycle
$8	1	Parallel Extension of Incremental Encoder	$0 = signed data $1 = unsigned data	-	-
$9	2	Triggered Time Base, frozen	0	Time Base Scale Factor	-
$A	2	Triggered Time Base, running	0	Time Base Scale Factor	-
$B	2	Triggered Time Base, armed	0	Time Base Scale Factor	-
$C	1	Incremental Encoder, no extension	$0 = signed data $1 = unsigned data	-	-
$D	3/5	Exponential and tracking filter of parallel data	$0 = exponential filter $8 = tracking filter	Max Change per Cycle	Filter Gain (Inverse Time Constant)
$E	-	Resolver conversion (Geo PMAC only)	$0 = clockwise $8 = counter-clockwise	-	-
$F	2	High-Resolution Interpolator	$0 = PMAC(1)-style ASIC $8 = PMAC2-style ASIC	Address of 1st A/D converter	-
$F	3/5	Hi-Res Interpolator (Geo PMAC only)	$0 = conversion $8 = diagnostic	Address of 1st A/D converter	sine/cosine bias

Incremental Encoder Entries ($0, $8, $C): These three conversion table methods utilize the incremental encoder registers in the Servo ICs. Each method provides a processed result with the units of (1/32) count – the low 5 bits of the result are fractional data.

1/T Extension: With the $0 method, the fractional data is computed by dividing the Time Since Last Count register by the Time Between Last 2 Counts register. This technique is known as 1/T extension, and is the default and most commonly used method. It can be used with a digital incremental encoder connected directly to the PMAC, either on PMAC(1) or PMAC2.

Parallel Extension: With the $8 method, the fractional data is computed by reading the five inputs at bits 19-2,3 either of the specified address (USERn, Wn, Vn, Un, and Tn flag inputs, respectively) in the case of a PMAC2, or of the specified address plus 4 (CHC[n+1], HMFL[n+1], +LIM[n+1], -LIM[n+1], FAULT[n+1]) in the case of a PMAC(1). This technique is known as parallel extension, and can be used with an analog incremental encoder processed through an ACC-8D Opt 8 Analog Encoder Interpolator board or its equivalent.

No Extension: In the $C method, the fractional data is always set to zero, which means there is no extension of the incremental encoder count. This setting is used mainly to verify the effect of one of the two extension methods. It is also recommended when feeding back the pulse-and-direction outputs for stepper drives.

With any of these three conversion methods, the source address in the low 16 bits (bits 0 – 15) is that of the starting register of the machine interface channel.

The first table below shows the entries for PMAC(1) encoder channels. The "m" in the first hex digit (bits 20 – 23) represents the conversion method ($0, $8, or $C). The "x" in the second hex digit represents a 0 (not summed) or 1 (summed with above entry).

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1	$mxC000	5	$mxC010	9	$mxC020	13	$mxC030
2	$mxC004	6	$mxC014	10	$mxC024	14	$mxC034
3	$mxC008	7	$mxC018	11	$mxC028	15	$mxC038
4	$mxC00C	8	$mxC01C	12	$mxC02C	16	$mxC03C

The next table below shows the entries for PMAC2 encoder channels. The "m" in the first hex digit (bits 20 – 23) represents the conversion method ($0, $8, or $C). The "x" in the second hex digit represents a 0 (not summed) or 1 (summed with above entry).

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1	$mxC000	5	$mxC020	9	$mxC040	13	$mxC060
2	$mxC008	6	$mxC028	10	$mxC048	14	$mxC068
3	$mxC010	7	$mxC030	11	$mxC050	15	$mxC070
4	$mxC018	8	$mxC038	12	$mxC058	16	$mxC078

ACC-28 Style A/D Entries ($1, $5): The "A/D" feedback entries read from the high 16 bits of the specified address and shift the data right three bits so that the least significant bit of the processed result in bit 5. Unlike the "parallel feedback" methods, this method will not "roll over" and extend the result.

The $1 method processes the information directly, essentially a copying with shift. The $5 integrates the input value as it copies and shifts it. That is, it reads the input value, shifts it right three bits, adds the bias term in the second line, and adds this value to the previous processed result.

If the bit 19 of the entry is '0' (making the second hex digit $0), the 16-bit source value is treated as a signed quantity; this should be used for the ACC-28A. If bit 19 of the entry is '1' (making the second hex digit $8), the 16-bit value is treated as an unsigned quantity; this should be used for the ACC-28B.

The first table shows the entry values that should be used for ACC-28 boards interfaced to PMAC(1) Servo ICs. The "m" in the first hex digit refers to the method digit – $1 for un-integrated; $5 for integrated. The "x" in the second digit is set to $0 for an ACC-28A signed A/D converter, or $8 for an ACC-28B unsigned A/D converter.

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1	$mxC006	5	$mxC016	9	$mxC026	13	mxC036
2	$mxC007	6	$mxC017	10	$mxC027	14	mxC037
3	$mxC00E	7	$mxC01E	11	$mxC02E	15	mxC03E
4	$mxC00F	8	$mxC01F	12	$mxC02F	16	mxC03F

The next table shows the entry values that should be used for ACC-28B boards interfaced to PMAC2 Servo ICs. . The "m" in the first hex digit refers to the method digit – $1 for un-integrated; $5 for integrated.

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1A	$m8C005	5A	$m8C025	9A	$m8C045	13A	$m8C065
1B	$m8C006	5B	$m8C026	9B	$m8C046	13B	$m8C066
2A	$m8C00D	6A	$m8C02D	10A	$m8C04D	14A	$m8C06D
2B	$m8C00E	6B	$m8C02E	10B	$m8C04E	14B	$m8C06E
3A	$m8C015	7A	$m8C035	11A	$m8C055	15A	$m8C075
3B	$m8C016	7B	$m8C036	11B	$m8C050	15B	$m8C076
4A	$m8C01D	8A	$m8C03D	12A	$m8C05D	16A	$m8C07D
4B	$m8C01E	8B	$m8C03E	12B	$m8C05E	16B	$m8C07E

Integration Bias: The $5 integrated format requires a second line to specify the bias of the A/D converter. This bias term is a signed quantity (even for an unsigned A/D converter), with units of 1/256 of the LSB of the 16-bit A/D converter. This value is *subtracted* from the reading of the ADC before the integration occurs.

For example, if there were an offset in a 16-bit ADC of +5 LSBs, this term would be set to 1280. If no bias is desired, a zero value should be entered here. If the conversion is unsigned, the result after the bias is not permitted to be less than zero. This term permits reasonable integration, even with an analog offset.

Parallel Feedback Entries ($2, $3, $6, $7) [Modified]: The "parallel feedback" entries read a word from the address specified in the low 16 bits (bits 0 – 15) of the first line, either a whole word from the single address specified, or three bytes from three consecutive Y-word addresses starting at the specified address. The four methods in this class, specified by the first hex digit (bits 20 – 23) are:

- $2: Y-word parallel, no filtering (2-line entry)
- $3: Y-word parallel, with filtering (3-line entry)
- $6: X-word parallel, no filtering (2-line entry)
- $7: X-word parallel, with filtering (3-line entry)

The second hex digit (bits 16 – 19) contains four mode-control bits that govern how the conversion is done. The four mode-control bits are:

- Bit 16 (digit value 1): Summing control (0 = no summing; 1 = sum with above result) for word-wide reads only; byte-select bit 0 for byte-wide reads only (depending on bit 18)
- Bit 17 (digit value 2): Reserved for word-wide reads, byte-select bit 1 for byte-wide reads
- Bit 18 (digit value 4): Shift-right/byte-wide-read control (depending on bits 16 and 17)
- Bit 19 (digit value 8): Shift-left control (0 = normal 5-bit left shift; 1 = no or right shift)

These four bits provide multiple combinations for the second hex digit, shown in the following table:

Second Digit	Conversion Details	Second Digit	Conversion Details
$0	Word-wide source, shift result left 5 bits, no summing	$8	Word-wide source, no shift of result, no summing
$1	Word-wide source, shift result left 5 bits, summed with above result	$9	Word-wide source, no shift of result, summed with above result
$2	(Reserved)	$A	(Reserved)
$3	(Reserved)	$B	(Reserved)
$4	(Reserved)	$C	Word-wide source, shift result right 3 bits, no summing
$5	Byte-wide Y source (low byte), shift result left 5 bits, no summing	$D	Byte-wide Y source (low byte), no shift of result, no summing
$6	Byte-wide Y source (middle byte), shift result left 5 bits, no summing	$E	Byte-wide Y source (middle byte), no shift of result, no summing
$7	Byte-wide Y source (high byte), shift result left 5 bits, no summing	$F	Byte-wide Y source (high byte), no shift of result, no summing

Shift control: With the "normal shift", the LSB of the source register is shifted to bit 5 of the result register, providing the standard 5 bits of (non-existent here) fractional position data. In this case, PMAC software regards the LSB as one "count" of position. With the "3-bit right shift", bit 8 of the source register is shifted to bit 5 of the result register. This is appropriate for 16-bit data found in the high 16 bits of the source register, such as the old MACRO Type 0 feedback.

With "no shift", the LSB of the source register ends up in bit 0 of the result register. This mode is used for one of three reasons:

- The data already comes with 5 bits of fraction, as from a MACRO Station.
- The normal shift limits the maximum velocity too much ($V_{max}<2^{18}$ LSBs per servo cycle)
- The normal shift limits the position range too much (Range$<\pm 2^{47}$/Ix08/32 LSBs)

Unless this is done because the data already contains fractional information, the "unshifted" conversion will mean that the motor position loop will consider 1 LSB of the source to be 1/32 of a count, instead of 1 count.

Word-Wide vs. Byte-Wide: Most types of position data read with the parallel conversion will be present in a single data word of up to 24 bits. This is "word-wide" data, read with the control bit in bit 18 set to 0, or bit 18 set to 1 and bits 16 and 17 both set to 0.

However, on some interface boards, such as the ACC-14P, the bus interface is only byte wide, so position data of more than 8 bits is read in bytes of consecutive registers. For this format, the control bit in bit 18 should be set to 1, and the combined value of bits 16 and 17 set to 1, 2, or 3.

If bits 16 and 17 set a combined value of 1, the low bytes (bits 0 – 7) of the selected registers are read. If bits 16 and 17 set a combined value of 2, the middle bytes (bits 8 – 15) of the selected registers are read. If bits 16 and 17 set a combined value of 3, the high bytes (bits 16 – 23) of the selected registers are read.

With the byte-wide read, right shifting of the result data is not supported, and summing of the result with the previous result is not supported. Only Y-registers can be read in byte-wide format, so byte-wide reads are supported only in methods $2 and $3, not $6 and $7.

In the byte-wide read, the low 16 bits (last 4 hex digits) of the first setup line specify the address of the first of the three Y-registers to be read. The Y-registers at the next two higher-numbered addresses will

also be read. The selected bytes of these three registers are combined into a single 24-bit value, with the selected byte of the first register forming the least significant byte of this value. The mask word of the second setup line (see below) then operates on this combined value as if it had come from a single 24-bit word.

Examples of this byte-wide conversion with the ACC-14P are shown below.

Mask Word: The second setup line is a 24-bit mask word that indicates which bits of the 24-bit word-wide source register, or of the 24-bit value combined from 3 byte-wide reads, are to be used. Each bit that is to be used takes a 1 in the mask word; each bit that is not to be used takes a 0 in the mask word. The mask word is combined with the contents of the source register or combined value with a bit-by-bit AND operation before the data is processed further. A correct mask word is necessary to ensure that all bits of the source to be used are used, that no bits that are not to be used are used, and to handle rollover of the source data properly.

For example, a mask word of $000FFF causes the low 12 bits of the source register to be used, $07FFFF causes the low 19 bits to be used, and $FFFFFF causes all 24 bits to be used.

Maximum Change Word: If the method character for a parallel read is $3 or $7, specifying "filtered" parallel read, there is a third setup line for the entry. This third line contains the maximum change in the source data in a single cycle that will be reflected in the processed result, expressed in LSBs per servo cycle. The filtering that this creates provides an important protection against noise and misreading of data. This number is effectively a velocity value, and should be set slightly greater than the maximum true velocity ever expected.

Common Parallel Data Sources: Any register can be read as a parallel data source, but the most common sources are MACRO feedback registers, MLDT timer registers, ACC-14D/V latched input registers, and ACC-14P byte-wide latched input registers. Each of these is covered below.

MACRO Feedback: When receiving position data over the MACRO ring with the "Type 1" protocol used in Delta Tau and most other MACRO devices, the position feedback appears in the 24-bit Register 0 for the "servo node". Servo nodes are mapped into Y-registers in PMAC2, and the MACRO protocol has its own error detection, so typically method $2 is used (Y-register, no filtering). This position data has usually already been processed in the encoder conversion table of the remote MACRO Station and comes back with five bits of fractional information, so it does not need to be shifted in the conversion table, making the second digit $8.

MACRO Type 1 Position Feedback

Node	1st Setup Line	Node	1st Setup Line	Node	1st Setup Line	Node	1st Setup Line
0	$28C0A0	4	$28C0A8	8	$28C0B0	12	$28C0B8
1	$28C0A4	5	$28C0AC	9	$28C0B4	13	$28C0BC

Sometimes a MACRO I/O node is used to bring back additional position data. I/O nodes are mapped into X-registers in PMAC2, so typically method $6 is used (X-register, no filtering). This position data also has typically been processed in the remote MACRO Station, so the second digit is $8 for no shifting here.

MACRO I/O Node Register 0 as Alternate Position Feedback

Node	1st Setup Line	Node	1st Setup Line	Node	1st Setup Line	Node	1st Setup Line
2	$68C0A0	6	$68C0A8	10	$68C0B0	14	$68C0B8
3	$68C0A4	7	$68C0AC	11	$68C0B4	15	$68C0BC

MLDT Feedback: PMAC2 Servo ICs have the ability to interface directly to magnetostrictive linear displacement transducers (MLDTs), outputting the excitation pulse, receiving the echo pulse, and

measuring the time between the two. This time is directly proportional to the distance. For this feedback the "time between last two counts" register is used like an absolute encoder. The following table shows the first line of the parallel feedback entry for each channel's timer register:

MLDT Timer Entries

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1	$30C000	5	$30C020	9	$30C040	13	$30C060
2	$30C008	6	$30C028	10	$30C048	14	$30C068
3	$30C010	7	$30C030	11	$30C050	15	$30C070
4	$30C018	8	$30C038	12	$30C058	16	$30C078

The second line in an MLDT entry should be $07FFFFF to specify the use of the low 19 bits.

The third line in an MLDT entry should contain a number slightly greater than the maximum velocity ever expected, expressed as timer increments per servo cycle. An increment of the 120 MHz timer represents about 0.024mm (0.0009 in) on a typical MLDT device. This value represents the maximum change in position reading that will be passed through the conversion table in a single servo cycle, and it provides an important protection against missing or spurious echo pulses.

Word-Wide Parallel Feedback: The ACC-14D and 14V boards are often used to connect parallel data from an absolute encoder or interferometer. The following table shows the entry first lines for the registers on these boards. The latched input registers on ACC-14D/V boards are mapped into Y-registers and filtering is usually desired, so a $3 method digit is used. In most cases, the normal shift is applied, making the second digit $0, but if the feedback has very high resolution, as can happen with an interferometer, the second digit should be set to $8 to disable shifting.

ACC-14D/V Port Entries

ACC 14 #	Port	Entry	ACC 14 #	Port	Entry
1	A	$3xFFD0	4	A	$3xFFE8
1	B	$3xFFD1	4	B	$3xFFE9
2	A	$3xFFD8	5	A	$3xFFF0
2	B	$3xFFD9	5	B	$3xFFF1
3	A	$3xFFE0	6	A	$3xFFF8
3	B	$3xFFE1	6	B	$3xFFF9

x = 0: normal shift; x = 8: no shift

Byte-Wide Parallel Feedback: The ACC-14P is a PCI-format board that can be used to connect parallel data from an absolute encoder or interferometer. The following table shows the entry first lines for the registers on these boards. The latched input registers on ACC-14D/V boards are mapped into Y-registers and filtering is usually desired, so a $3 method digit is used.

The data is byte-wide, so bit 18 (value of 4 in the second digit) is set to 1. Ports A and B occupy the low bytes of each word, so bits 16 and 17 get a combined value of 1, making the second hex digit $5. Ports C and D occupy the middle bytes of each word, so bits 16 and 17 get a combined value of 2, making the second hex digit $6. If no shifting of the result data is desired, bit 19 (value of 8 in this digit) is also set to 1, making the second hex digit $D or $E

Ports A and C on these boards start in the board's base address; Ports B and D start in {Base+3}.

ACC-14P Port Entries

ACC 14 #	Port	Entry	ACC 14 #	Port	Entry
1	A	$35FFD0	4	A	$35FFE8
1	B	$35FFD3	4	B	$35FFEB
1	C	$36FFD0	4	C	$36FFE8
1	D	$36FFD3	4	D	$36FFEB
2	A	$35FFD8	5	A	$35FFF0
2	B	$35FFDB	5	B	$35FFF3
2	C	$36FFD8	5	C	$36FFF0
2	D	$36FFDB	5	D	$36FFF3
3	A	$35FFE0	6	A	$35FFF8
3	B	$35FFE3	6	B	$35FFFB
3	C	$36FFE0	6	C	$36FFF8
3	D	$36FFE3	6	D	$36FFFB

Time-Base Entries ($4, $9, $A, $B): A time-base entry performs a scaled digital differentiation of the value in the source register. It is most often used to perform "electronic cam" functions, slaving a motion sequence to the frequency of a master encoder. There are two types of time-base entries: untriggered and triggered. An untriggered time base does not provide a specific starting point in the master source data. A triggered time base starts the differentiation upon receipt of a hardware trigger on the master encoder's channel, referenced to the position captured by that trigger. This can be used to create an absolute synchronization between the master position and the slave trajectory.

Time-base entries are two-line entries. The first setup line contains the method digit and the address of the source-data register. The second setup line contains the "time-base scale factor". The first result line contains the intermediate result value of the source data, saved for the next cycle to be able to compute the differentiation. The second result line contains the final result, which is the differentiated value. Most commonly this result is used as the time-base source for a coordinate system, so Ix93 for the coordinate system points to this second line.

Untriggered Time Base ($4): In an untriggered time-base entry, the first setup line contains a "4" in the method digit (bits 20 – 23) and the address of the source register in bits 0 – 15. The source register is almost always the result register of an incremental encoder entry (e.g. 1/T) higher in the table (addresses $0720 to $072D). For example, to use the result of the fourth line of the conversion table as a source, this I-variable would be $400723.

The second setup line is the "time-base scale factor" which multiplies the differentiated source value. The final result value equals *2 * Time-Base-Scale-Factor * (New Source Value - Old Source Value)*. "New Source Value" and "Old Source Value" (stored from the previous servo cycle) are typically in units of 1/32 of a count, the usual scaling of a 1/T encoder conversion result.

When this time base entry is used to calculate a frequency-based time base for a coordinate system, the TBSF should be set to 2^{17}/Real-Time Input Frequency (131,072/RTIF), where the Real-Time Input Frequency (RTIF) in counts per millisecond, is the frequency at which motion trajectories using this time base will execute at the programmed speed or in the programmed time. The motion sequence to be slaved to this frequency should be written assuming that the master is always generating this real-time input frequency (so always moving at the "real-time speed"). The true speed of trajectories using this time base will vary proportionately with the actual input frequency.

Example

The application requires the use of Encoder 4 on board a PMAC2 as an untriggered time-base master for Coordinate System 1. The real-time input frequency is selected as 256 counts/msec. The conversion table starts with 8 single-line entries in Y:$0720 – Y:$0727, with the 4th line (Y:$0723) doing a 1/T conversion of Encoder 4.

```
; Setup on-line commands
WY:$0723,$00C018  ; 1/T conversion of Encoder 4
WY:$0729,$400723  ; Unriggered time base from 1/T encoder
WY:$072A,512      ; TBSF=131072/256
I193=$072A        ; C.S.1 use result for time base
```

Triggered Time Base ($9, $A, $B): A "triggered" time-base entry is like a regular untriggered time-base entry, except that it is easy to freeze the time base, then start it exactly on receipt of a trigger that captures the "starting" master position or time.

In a triggered time-base entry, the first setup line (I-variable) contains a 9, A, or B in the method digit (bits 20 – 23), depending on its present state. It contains the address of the source register in bits 0 – 15. The source register for triggered time base must be the starting (X) address for one of the machine interface channels of a Servo IC.

The second setup line (I-variable) is the time-base scale factor which multiplies the differentiated source value. The final result value (when running) equals *64 * Time-Base-Scale-Factor * (New Source Count - Old Source Count)*. New Source Count and Old Source Count are the values of the addressed encoder counter, in whole counts.

When this time-base entry is used to calculate a frequency-based time base for a coordinate system, the TBSF should be set to 2^{17}/Real-Time Input Frequency (131,072/RTIF), where the Real-Time Input Frequency (RTIF) in counts per millisecond, is the frequency at which motion trajectories using this time base will execute at the programmed speed or in the programmed time. The motion sequence to be slaved to this frequency should be written assuming that the master is always generating this real-time input frequency (so always moving at the "real-time speed"). The true speed of trajectories using this time base will vary proportionately with the actual input frequency.

A triggered time-base entry in Turbo PMAC automatically computes the 1/T count extension of the input frequency itself before the differentiation. It computes this to 1/32 of a count.

In use, the method digit (comprising bits 20-23 of the first line) is changed as needed by setting of the I-variable. Triggered time base has three states, frozen, armed, and running, all of which must be used to utilize the triggering feature.

First, the method digit is set to $9 (e.g. **`WY:$0728,$90C00C`**, or **`M190=$9`** with **`M190->Y:$0728,20,4`**) before the calculations of the triggered move are started, to freeze the time base (and therefore the motion) while the move calculations are done. This is typically done in the user's motion program. When this entry is in the frozen state, the table reads the channel's captured position register each servo cycle to ensure the triggering logic is reset for the next capture. The final result of the entry is always 0 when frozen.

> ***Note:***
>
> In a PMAC application with a fast CPU and a light computational load, it is possible that the entry will not be in the "frozen" state during a servo interrupt, and the table will not get a chance to reset the trigger logic. Therefore, it is advisable to reset the triggering logic explicitly in the user program with a "dummy" read of the channel's captured position register, which is the X-register with an address 3 greater than the address specified in the entry (e.g. X:$C00B if the entry specifies $C008). The suggested M-variable for the captured position register is Mxx03.

Next, the method digit is set to $B (e.g. **`WY:$0728,$B0C00C`**, or **`M190=$B`** with **`M190->Y:$0728,20,4`**) after the calculations of the triggered move are finished, to arm the time base for the trigger. This is typically done in a PLC program that simply looks to see if the entry is frozen and changes it to the armed state. The final result of the entry is always 0 when armed.

In the armed state, the ECT checks every servo cycle for the channel's trigger bit to be set. When the ECT sees the trigger (the capture trigger for the machine interface channel as defined by Encoder I-variable 2 and 3 for the channel used (e.g. I917 and I918 for a PMAC(1) channel 4 or I942 and I943 for a PMAC2 channel 4), it automatically sets the method digit to $A for "running" time base. It uses the position captured by the trigger as the starting position ("time zero") for the running time base.

The following tables show the possible first-line entries for triggered time base (running mode):

Triggered Time-Base Entries for PMAC(1)-Style Servo ICs (Running State)

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1	$A0C000	5	$A0C010	9	$A0C020	13	$A0C030
2	$A0C004	6	$A0C014	10	$A0C024	14	$A0C034
3	$A0C008	7	$A0C018	11	$A0C028	15	$A0C038
4	$A0C00C	8	$A0C01C	12	$A0C02C	16	$A0C03C

Triggered Time-Base Entries for PMAC2-Style Servo ICs (Running State)

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1	$A0C000	5	$A0C020	9	$A0C040	13	$A0C060
2	$A0C008	6	$A0C028	10	$A0C048	14	$A0C068
3	$A0C010	7	$A0C030	11	$A0C050	15	$A0C070
4	$A0C018	8	$A0C038	12	$A0C058	16	$A0C078

Example

The application requires the use of Encoder 4 on board a Turbo PMAC2 as a triggered time base master for coordinate system 1. It is to be triggered by the rising edge of its index channel. The real-time input frequency is selected as 256 counts/msec. The conversion table starts with 8 single-line entries in Y:$0720 – Y:$0727.

```
; Setup on-line command
WY:$0728,$A0C018  ; Triggered time base from PMAC2 channel 4
WY:$0729,512      ; TBSF=131072/256
I942=1            ; Channel 4 trigger on rising index
I193=$0729        ; C.S.1 use result for time base
M190->Y:$0728,20,4      ; Method digit of time base entry
M403->X:C01B,0,24,S     ; Channel's captured position register

; Motion program segment
DWELL 0           ; Stop any lookahead
M190=$9           ; Freeze the time base
P403=M403         ; Dummy read to ensure capture logic reset
X10               ; Calculate first move

; PLC program segment
IF (M190=$9)      ; If frozen
  M190=$B         ; Then arm
ENDIF
```

Low-Pass Filter Entries ($D): The $D entry is used to create one of two types of low-pass filters on a word of input data to provide smoothing of noisy measurements. The two types of filter are distinguished by bit 19 of the first setup line of the entry. If bit 19 is 0, making the second hex digit $0, the filter is a simple exponential filter. If bit 19 is 1, making the second hex digit $8, the filter is a more sophisticated tracking filter that includes an integrator to eliminate steady-state errors.

The simpler exponential filter, which is a three-line entry in the table, is suitable for the smoothing of noisy master data used for electronic gearing (position following) or electronic cams (external time base). However, it will produce lags even in the steady state (e.g. at constant velocity), so it is usually not suitable for smoothing servo feedback data because of the delays it introduces.

The more complex tracking filter, which is a five-line entry in the table, is suitable for smoothing either master data or feedback data, because its integrator eliminates steady-state errors. Still, its filtering can introduce delays in responding to dynamic changes (e.g. accelerations), so it needs to be set up carefully. This software tracking filter is dynamically equivalent to the hardware tracking filters common in resolver-to-digital converter ICs. It is commonly used to smooth the results of direct conversion of sinusoidal encoders and resolvers.

Exponential Filter ($D0xxxx)

The equation of the exponential filter executed every servo cycle *n* is:

$$If\ [In(n) - In(n-1)] > Max\_change,\ In(n) = In(n-1) + Max\_change$$
$$If\ [In(n) - In(n-1)] < -Max\_change,\ In(n) = In(n-1) - Max\_change$$
$$Out(n) = Out(n-1) + (K/2^{23})*[In(n)-Out(n-1)]$$

In, *Out*, and *K* are all signed 24-bit numbers (range -8,388,608 to 8,388,607). The difference *[In(n)-Out(n-1)]* is truncated to 24 bits to handle rollover properly.

The time constant of the filter, in servo cycles, is $(2^{23}/K)-1$. The lower the value of K, the longer the time constant.

No shifting action is performed. Any operations such as 1/T interpolation should have been done on the data already, so the source register for this filter is typically the result register of the previous operation.

Method/Address Word: The first setup line of an exponential filter entry contains a 'D' in the first hex digit (bits 20 – 23), a '0' in the second hex digit, and the address of the source X-register in the third through sixth hex digits (bits 0 – 15). If it is desired to execute an exponential filter on the contents of a Y-register, the contents of the Y-register must first be copied to an X-register in the conversion table with a "parallel" entry ($2) higher in the table. The source addresses for exponential filter entries are almost always from the conversion table itself (X:$0720 – X:$073C). For example, to perform an exponential filter on the result of the fourth line of the table, the first setup line of the filter entry would be $D00723.

Maximum Change Word: The second setup line of an exponential filter entry contains the value "max change" that limits how much the entry can change in one servo cycle. The units of this entry are whatever the units of the input register are, typically 1/32 of a count. For example, to limit the change in one servo cycle to 64 counts with an input register in units of 1/32 count, this third line would be 64*32 = 2048.

Filter Gain Word: The third setup line of an exponential filter entry contains the filter gain value *K*, which sets a filter time constant T_f of $(2^{23}/K)-1$ servo cycles. Therefore, the gain value *K* can be set as $2^{23}/(T_f+1)$. For example, to set a filter time constant of 7 servo cycles, the filter gain word would be 8,388,608/(7+1) = 1,048,576.

Result Word: The output value of the exponential filter is placed in the X register of the third line of the conversion table entry. An operation that uses this value should address this third register; for example Ix05 for position following, or the source address for a time-base conversion-table entry (to keep position lock in time base, this filter must be executed *before* the time-base differentiation, not afterward).

Resolver Conversion Entries ($E) [Geo PMAC only]: The $E entry converts the sine and cosine resolver feedback values processed through the Geo PMAC's A/D converter (ADC) registers to a 14-bit resolver angle value.

Method/Address Word: The first setup line of a resolver conversion entry contains $E in the first hex digit and the Y-address of the first ADC register to be read in the low 16 bits (the third through sixth hex digits). The next ADC register is read at the next higher Y-address. If bit 19 of the line is set to 0

(making the second hex digit $0) the conversion creates a "clockwise" rotation sense. If bit 19 of the line is set to 1 (making the second hex digit $8), the conversion creates a "counter-clockwise" rotation sense.

The two base ADC addresses presently supported by the Geo PMAC for resolver conversion are Y:$FF00 for Channel 1 and Y:$FF20 for Channel 2. Therefore, the possible first-setup-line values are:

First Setup Line	Conversion
$E0FF00	Channel 1 CW
$E8FF00	Channel 1 CCW
$E0FF20	Channel 2 CW
$E8FF20	Channel 2 CCW

Excitation Address Setup Word: The second setup line in a resolver conversion entry contains the address of the excitation value register in the low 16 bits (the third through sixth hex digits), used to correlate the excitation and the feedback values. The excitation register is presently at a fixed address of $FF5C in the Geo PMAC, so this line should be $00FF5C.

Sine/Cosine Bias Setup Word: The third setup line in a resolver conversion entry contains bias-correction terms for the sine and cosine ADC values. The high twelve bits (the first three hex digits) contain the bias-correction term for the sine input; the low twelve bits (the last three hex digits) contain the bias-correction term for the cosine input. Each 12-bit section should be treated as a signed 12-bit value (so if the most significant of the 12 bits is a 1, the bias value is negative).

Each 12-bit bias-correction term should contain the value opposite that which the high 12 bits of the matching A/D converter report when they should ideally report zero. In action, the bias term will be added to the high 12 bits of the corresponding ADC reading before subsequent calculations are done.

For example, if the bias-correction word were set to $004FFA, the sine bias correction would be +4 LSBs of a 12-bit ADC, and the cosine bias correction would be -6 LSBs ($FFA = -6) of a 12-bit ADC. In use, 4 12-bit LSBs would be *added to* the sine reading, and 6 12-bit LSBs would be *subtracted from* the cosine reading each cycle before further processing.

In most cases, the bias-correction word will be determined automatically by an analog "diagnostic" entry in the conversion table. The result of that diagnostic entry, containing both bias corrections, can simply be copied into this setup word.

The resolver conversion can only be used if the Geo PMAC's Feedback Option 1 for analog position feedback is ordered.

Result Word: The output value of the resolver conversion is placed in the 24-bit X-register of the third line of the conversion table entry. The values in bits 5 – 16 of the result word contain the high 12 bits of the calculated arctangent of the bias-corrected sine and cosine values from the resolver. Because PMAC software considers the value in bit 5 to be a "count" for its scaling purposes, this conversion returns resolver position values of a 12-bit conversion (4096 "counts" per cycle of the resolver).

However, because the conversion uses dual 14-bit converters and the arctangent calculations compute more than 12 bits, the result contains additional resolution in bits 0 – 4 that PMAC software considers to have "fractional", but still real, count resolution. If the electromagnetic noise levels are low, and the signals use near the full scale of the ADCs, a repeatable 14-bit resolution (16,384 states per cycle of the resolver) can be achieved.

Bits 17 – 23 of the result contain cycle data from software extension of the result to multiple resolver cycles. If the result is then used for feedback or master data, it will be further extended in the motor algorithms.

This resolver conversion is a direct, and not a tracking, conversion. As such, it is more dynamically responsive, but also more susceptible to measurement noise. If a more noise-immune result is desired, at the cost of some dynamic responsiveness (but still no steady-state tracking errors), a digital tracking filter can be implemented on this result with another conversion table entry (format $D8). The result of that filter entry can then be used as the feedback or master data.

High-Resolution Interpolator Entries ($F): An ECT entry in which the first hex digit of the first line is $F processes the result of a high-resolution interpolator for analog "sine-wave" encoders, such as the ACC-51. This entry, when used with a high-resolution interpolator, produces a value with 4096 states per line. The entry must read both an encoder channel for the whole number of lines of the encoder, and a pair of A/D converters to determine the location within the line, mathematically combining the values to produce a single position value.

Encoder Channel Address: The first line of the two-line entry contains $F in the first hex digit and the base address of the encoder channel to be read in the low 16 bits (bits 0 to 15). If bit 19 of the line is set to 0 (making the second hex digit $0), PMAC expects a PMAC(1)-style Servo IC on the interpolator, as in the ACC-51P. If bit 19 is set to 1 (making the second hex digit $8), PMAC expects a PMAC2-style Servo IC for the interpolator, as in the ACC-51S for the PMAC2A-PC/104.

The following table shows the possible entries when PMAC(1)-style Servo ICs are used, as in the ACC-51P.

High-Res Interpolator Entry First Lines for ACC-51P

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
9	$F0C020	11	$F0C028	13	$F0C030	15	$F0C038
10	$F0C024	12	$F0C02C	14	$F0C034	16	$F0C03C

The next table shows the possible entries for the ACC-51S, which uses the Servo ICs of the PMAC2A-PC/104 main board and the ACC-1P Axis 5-8 board:

High-Res Interpolator Entry First Lines for ACC-51S

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1	$F8C000	3	$F8C010	5	$F8C020	7	$F8C030
2	$F8C008	4	$F8C018	6	$F8C028	8	$F8C038

A/D Converter Address: The second line of the entry contains $00 in the first two hex digits and the base address of the first of two A/D converters to be read in the low 16 bits (bits 0 to 15, the last four hex digits). The second A/D converter will be read at the next higher address. The following table shows the possible entries when the ACC-51P is used:

High-Res Interpolator Entry Second Lines for ACC-51P

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
9	$00C022	11	$00C02A	13	$00C032	15	$00C03A
10	$00C026	12	$00C02E	14	$00C036	16	$0FC03E

The next table shows the possible entries when the ACC-51S is used:

High-Res Interpolator Entry Second Lines for ACC-51S

Channel	Entry	Channel	Entry	Channel	Entry	Channel	Entry
1	$00FFC0	3	$00FFC4	5	$00FFC8	7	$00FFCC
2	$00FFC2	4	$00FFC6	6	$00FFCA	8	$00FFCE

ResultWord: The output value of the high-resolution sinusoidal encoder conversion is placed in the 24-bit X-register of the second line of the conversion table entry. Bit 0 of the result contains the LSB of the

conversion representing 1/4096 of a line of the encoder. Since PMAC software considers Bit 5 to be a "count" for scaling purposes when used for servo feedback or master data, Bit 0 will be considered 1/32 of a count. This means that PMAC software will scale the data as 128 "software counts" per line of the encoder.

High-Resolution Encoder Interpolation Entries ($F) [New 3-line or 5-line entry for Geo PMAC only]: The $F entry in the Geo PMAC is used to process the feedback from sinusoidal incremental encoders through the Geo PMAC's high-resolution interpolation circuitry in one of two ways. The two methods are distinguished by bit 19 of the first setup line of the entry. If bit 19 is 0, making the second hex digit $0, it is a 3-line entry that simply produces a position result with 4096 states per line of the encoder, suitable for direct use as feedback or master data. If bit 19 is 1, making the second hex digit $8, it is a 5-line entry that produces several data results useful for setup and diagnostics of the feedback. Note that this conversion type operates differently from the interpolation entries on other types of PMACs.

High-Resolution Interpolation Position Entry ($F0xxxx) [Geo PMAC only]

If bit 19 of the first setup line of a Geo PMAC interpolation entry is 0, to specify an entry that produces a usable position result, this is a three-line entry (as opposed to two lines on other PMACs). The entry combines whole-line information from the encoder counter whose address is specified in the first line with fractional-line information from the arctangent of the A/D converters whose address is specified in the second line. Note that the direction sense of both parts must agree. On power-up/reset, PMAC checks the direction sense of the counter and matches the direction sense of the fractional information to this. However, if the direction sense of the counter is then changed with the encoder-decode I-variable, the new settings must be saved and the PMAC reset to restore proper direction mapping.

Method/Address Setup Word: The first setup line of the three-line entry contains $F in the first hex digit, $0 in the second hex digit, and the base address of the encoder channel to be read in the low 16 bits (the third through sixth hex digits). In the Geo PMAC, the first encoder channel is at address $C000 and the second encoder channel is at address $C008, so the first setup line is set to $F0C000 or $F0C008.

A/D-Converter Address Setup Word: The second line of the entry contains $00 in the first two hex digits and the address of the first of the two A/D converters in the low 16 bits (the last four hex digits). The second A/D converter will be read at the next higher address. In the Geo PMAC, the first A/D converter for Channel 1 is at address $FF00, and the first A/D converter for Channel 2 is at address $FF20, so the second setup line is set to $00FF00 or $00FF20.

Sine/Cosine Bias Setup Word: The third setup line in a high-resolution sinusoidal-encoder conversion entry contains bias-correction terms for the sine and cosine ADC values. The high twelve bits (the first three hex digits) contain the bias-correction term for the sine input; the low twelve bits (the last three hex digits) contain the bias-correction term for the cosine input. Each 12-bit section should be treated as a signed 12-bit value (so if the most significant of the 12 bits is a 1, the bias value is negative).

Each 12-bit bias-correction term should contain the value opposite that which the high 12 bits of the matching A/D converter report when they should ideally report zero. In action, the bias term will be added to the high 12 bits of the corresponding ADC reading before subsequent calculations are done.

For example, if the bias-correction word were set to $004FFA, the sine bias correction would be +4 LSBs of a 12-bit ADC, and the cosine bias correction would be -6 LSBs ($FFA = -6) of a 12-bit ADC. In use, 4 12-bit LSBs would be *added to* the sine reading, and 6 12-bit LSBs would be *subtracted from* the cosine reading each cycle before further processing.

In most cases, the bias-correction word will be determined automatically by an analog "diagnostic" entry in the conversion table. The result of that diagnostic entry, containing both bias corrections, can simply be copied into this setup word.

Result Word: The output value of the high-resolution sinusoidal-encoder conversion in the Geo PMAC is placed in the 24-bit X-register of the third line of the conversion table entry. Bit 0 of the result contains the LSB of the conversion, representing 1/4096 of a line of the encoder. Since PMAC software considers the contents of Bit 5 to be a "count" for scaling purposes when used for servo feedback or master data, bit 0 will be considered 1/32 of a count. This means that PMAC software will scale the data as 128 "software counts" per line of the encoder.

High-Resolution Interpolation Diagnostic Entry ($Fxxxxx, bit 19 = 1) [Geo PMAC only]

If bit 19 of the first setup line of a Geo PMAC "interpolation" entry is 1, to specify an entry that produces either vector magnitude or analog-input bias terms for the sine and cosine inputs of a sinusoidal encoder or resolver, this is a five-line entry. These result values can be used to verify proper setup and interface of the encoder or resolver and to optimize the accuracy of the conversion during initial setup, and/or to check for loss of the encoder or resolver during the actual application. Bit 16 of the first setup line determines whether the result produced is the sum of the squares of the two analog inputs (bit 16 = 0) or the bias terms for the analog inputs (bit 16 = 1).

Method/Address Setup Word: The first setup line of the five-line entry contains $F in the first hex digit, $8 in the second hex digit (bit 19 = 1) to produce a sum-of-squares result or $9 in the second digit (bit 19 = 1, bit 16 =1) to produce a bias correction term, and the address of the first of the two A/D converters in the low 16 bits (the last four hex digits). The second A/D converter will be read at the next higher address. In the Geo PMAC, the first A/D converter for Channel 1 is at address $FF00, and the first A/D converter for Channel 2 is at address $FF20, so the first setup line is set to $F8FF00 or $F8FF20 to produce a sum-of-squares result, or to $F9FF00 or $F9FF20 to produce a bias-correction result.

When set up to determine the bias term, if bit 17 is set to 0, making the second hex digit $9 (as the above instructions say), the minimum and maximum values for the sine and cosine readings that are used to determine the appropriate bias values are set to 0. As soon as the Geo PMAC starts accumulating minimum and maximum values (the next servo cycle), it will set bit 17 to 1, making the second hex digit to $B. If you want to start a new test, for example after a circuit adjustment, you must set bit 17 to 0 again by making the second hex digit $9.

Reserved Setup Word: The second setup line of this entry type is reserved for future use, and should be left at 0.

Active Bias Correction Setup Word: The third setup line of the five-line entry contains the sine and cosine bias terms that are used in the sum-of-squares calculations. Two signed 12-bit bias terms are combined in a 24-bit word. The sine bias-correction term is in the high 12 bits (bits 12 – 23); the cosine bias-correction term is in the low 12 bits (bits 0 – 11). These terms match the high 12 bits from the corresponding A/D converters. This word does not necessarily match the bias "result" term derived from using this entry to determine a suggested bias correction, or the bias correction used in the "feedback" table entry for the encoder or resolver.

Reserved Setup Words: The fourth and fifth setup lines of this entry type are reserved for future use, and should be left at 0.

Result Word (Sum of Squares): When bit 16 of the first setup line is 0, the final (fifth) result word contains the sum of squares of the biased sine and cosine measurements for the most recent servo cycle.

$$Result = (SineADC + SineBias)^2 + (Cosine\ ADC + CosineBias)^2$$

The values *SineADC* and *CosineADC* are read from the A/D converters at the address specified in the first setup line. The values *SineBias* and *CosineBias* are read from the third setup line.

To understand the scaling of the result word, it is best to think of all four of the values as being normalized, that is, as having a valid range of -1.0 to +1.0. With small bias terms, the sum of squares

result would have a possible normalized value of 0.0 to +2.0. When read as an unsigned integer, this register has a range of 0 to 16,777,215 ($FFFFFFF), corresponding to a normalized range of 0.0 to 2.0.

When the encoder and interpolator circuitry, or the resolver and excitation circuitry, are working properly, the sum of squares should have a normalized value of +0.25 to +0.9999 (2,097,152 to 8,388,607, or $200000 to $7FFFFF). If the resulting normalized value is greater than or equal to +1.0 (8,388,608, or $800000), meaning that the most significant bit (bit 23) is set to 1, at any point in the cycle, this indicates that saturation has occurred in at least one of the readings due to either too large a signal or a significant bias. This should be corrected before using this sensor in actual operation.

If the result has a normalized value of less than +0.25 (2,097,152, or $200000), meaning that bits 23, 22, and 21 are all 0, at low sensor frequencies, the signals are too small to get full resolution from the result, and this should be corrected before using this sensor in actual operation. Many sinusoidal encoders do have a reduction in signal magnitude of up to one-half at their highest frequencies, reducing the magnitude of this square term by three-quarters, and this is acceptable.

It is possible to monitor this term in the actual application to check for loss of the encoder. If the inputs are no longer driven externally, for example because the cable has come undone, the positive and negative input pair to the ADC will pull to substantially the same voltage, and the output of the ADC will be a very small number, resulting in a small magnitude of the sum of squares in at least part of the cycle. (If both signals cease to be driven externally, the sum of squares will be small over the entire cycle). The high four bits (bits 20 – 23) of the sum-of-squares result can be monitored, and if the four-bit value goes to 0, it can be concluded that the encoder has been "lost", and the motor should be "killed".

Ideally, the magnitude of the sum-of-squares result should be constant throughout the sine/cosine cycle, at least at constant frequency. If there is significant variation, this is an indication of signal imperfection. In most cases, the most important imperfection is a DC bias on the sine and/or cosine signals. This entry can be used in its alternate format to determine the optimal bias correction. Once that bias correction has been determined (the result word in that format), it can be copied into the active correction setup word for the diagnostic entry, and the entry put back into sum-of-squares mode, as an important verification that a good bias correction has been determined.

A/D Bias Result Word: When bit 16 of the first setup line is 1, the final (fifth) result word contains the suggested bias correction word containing the bias correction terms for the sine and cosine terms. This 24-bit value, containing two signed 12-bit correction terms, can be copied into the third setup word for the interpolator diagnostic entry for confirmation of its effect, and to the third line of the interpolator feedback entry, or the resolver feedback entry, for actual use. The sine bias-correction term is in the high 12 bits (bits 12 – 23); the cosine bias-correction term is in the low 12 bits (bits 0 – 11).

In this mode, the encoder should be moved for several seconds (motion by hand is OK) to ensure good sampling of maximums and minimums of both waveforms and accurate bias-correction terms. It is probably best to do this test with the amplifier disabled to prevent the possibility of noise distorting the maximum and minimum readings.

VME/DPRAM Addressing Setup Registers: X:$0783 – X:$078C

There are ten saved setup registers that configure the VME bus interface, including DPRAM. In ISA-bus PMACs, two of these registers configure the optional DPRAM interface.

X:$0783 VME Address Modifier

Range: $00 - $FF

Units: None

Default: $39

X:$0783 controls which address modifier value PMAC will respond to when sent by the VME bus host. X:$0783 takes one of three valid values in normal use, depending on the address bus width used:

- X:$0783 = $29: 16-bit addressing
- X:$0783 = $39: 24-bit addressing
- X:$0783 = $09: 32-bit addressing

X:$0783 is actually used at power-on/reset only, so to set or change the VME address modifier, change the value of X:$0783, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$0783 is copied at power-on/reset is X:$E006 bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M90) to change the active setup without a **SAVE** and reset.

X:$0784 VME Address Modifier Don't Care Bits

Range: $00 - $FF

Units: None

Default: $04

X:$0784 controls which bits of the X:$0783 VME address modifier are don't care bits. X:$0784 is set to $04 in all normal use, which permits both "non-privileged" and "supervisory" data access by the VME host.

X:$0784 is actually used at power-on/reset only, so to set or change the VME address modifier "don't care" bits, change the value of X:$0784, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$0784 is copied at power-on/reset is X:$E007 bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M91) to change the active setup without a **SAVE** and reset.

X:$0785 VME Base Address Bits A31-A24

Range: $00 - $FF

Units: None

Default: $FF

X:$0785 controls bits A31 through A24 of the VME bus base address of PMAC, both for the mailbox registers, and the dual-ported RAM. It is only used if 32-bit addressing has been selected with X:$0783 and X:$078C.

X:$0785 is actually used at power-on/reset only, so to set or change bits 16-23 of the VME bus base address, change the value of X:$0785, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$0785 is copied at power-on/reset is X:$E008 bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M92) to change the active setup without a **SAVE** and reset.

X:$0786 VME Mailbox Base Address Bits A23-A16 ISA DPRAM Base Address Bits A23-A16

Range: $00 - $FF

Units: None

Default: $7F (VME); $0D (ISA)

On VME bus systems, X:$0786 controls bits A23 through A16 of the VME bus base address of the mailbox registers for PMAC. Bit 7 of X:$0786 corresponds to A23 of the base address, and bit 0 of X:$0786 corresponds to A16. X:$0786 is only used on VME systems if 24-bit or 32-bit addressing has been selected with X:$0783 and X:$078C.

On ISA bus systems (PC, PC Ultralite), X:$0786 controls bits A23 through A16 of the ISA bus base address of the DPRAM. Bit 7 of X:$0786 corresponds to A23 of the base address, and bit 0 of X:$0786 corresponds to A16. A23 through A20 are only used on ISA bus systems if bit 2 of X:$0787 is set to 1, enabling 24-bit addressing.

X:$0786 is actually used at power-on/reset only, so to set or change the base address, change the value of X:$0786, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$0786 is copied at power-on/reset is X:$E009 bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M93) to change the active setup without a **SAVE** and reset.

X:$0787 VME Mailbox Base Address Bits A15-A08 ISA DPRAM Base Address Bits A15-A14 & Control

Range: $00 - $FF

Units: None

Default: $A0 (VME); $45 (ISA)

On VME bus systems, X:$0787 controls bits A15 through A08 of the VME bus base address of the mailbox registers of PMAC. Bit 7 of X:$0786 corresponds to A23 of the base address, and bit 0 of X:$0786 corresponds to A16. X:$0787 is used whether 16-bit, 24-bit, or 32-bit addressing has been selected with X:$0783 and X:$078C.

On ISA bus systems (PC and PC Ultralite), X:$0787 controls the enable state and addressing mode of the DPRAM. If the DPRAM is to appear as a 16k block of memory on the ISA bus, it also sets bits A15 and A14 of the ISA bus base address.

The first hex digit of X:$0787 contains bits 4 – 7. When the DPRAM is addressed as a 16k x 8 block of memory on the ISA bus, bit 7 of X:$0787 corresponds to A15, and bit 6 of X:$0787 corresponds to A14. Bits 5 and 4 must be set to 0.

The second hex digit of X:$0787 contains bits 0 – 3. These are individual control bits. Bits 0 and 2 control the addressing mode and block size. Bits 1 and 3 control the bank selection if the large DPRAM is addressed as a small block of memory. These should always be set to 0 in a non-Turbo PMAC. The commonly used settings of the second hex digit of X:$0787 are:

- 0: DPRAM not enabled
- 1: 20-bit addressing (below 1M), 16k x 8 address block
- 5: 24-bit addressing (above or below 1M), 16k x 8 address block

X:$0787 is actually used at power-on/reset only, so to set or change, and keep, these settings, change the value of X:$0787, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$0787 is copied at power-

on/reset is X:$E00A bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M94) to change the active setup without a **SAVE** and reset.

X:$0788 VME Interrupt Level

Range: $01 - $07

Units: None

Default: $02

X:$0788 controls which interrupt level (1 to 7) PMAC will assert on the VME bus. Multiple boards on the same VME bus may assert the same interrupt level if each one has a unique set of interrupt vectors as set by X:$0789.

X:$0788 is actually used at power-on/reset only, so to set or change the VME interrupt level, change the value of X:$0788, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$0788 is copied at power-on/reset is X:$E00B bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M95) to change the active setup without a **SAVE** and reset.

X:$0789 VME Interrupt Vector

Range: $00 - $FF

Units: None

Default: $A1

X:$0789 controls which interrupt vectors will be provided when PMAC asserts a VME bus interrupt. If PMAC asserts the interrupt to signify that it has read a set of mailbox registers and is ready to accept another set, the interrupt vector value will be equal to (X:$0789-1). If PMAC asserts the interrupt to signify that it has written to a set of mailbox registers and is ready for the host computer to read these, the interrupt vector value will be equal to X:$0789. If PMAC asserts the interrupt to signify that it has put a line of text in the DPRAM ASCII response buffer and is ready for the host computer to read this, the interrupt vector value will be equal to (X:$0789+1).

If there are multiple PMAC boards asserting the same interrupt level in the VME bus as set by X:$0788, they each must assert a unique, non-overlapping set of interrupt vectors.

X:$0789 is actually used at power-on/reset only, so to set or change the VME interrupt vector, change the value of X:$0789, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$0789 is copied at power-on/reset is X:$E00C bits 0 – 7.

It is permissible to write to this register directly (suggested M-variable M96) to change the active setup without a **SAVE** and reset.

X:$078A VME DPRAM Base Address Bits A23-A20

Range: $00 - $FF

Units: None

Default: $00

X:$078A controls bits A23 through A20 of the VME bus base address of the dual-ported RAM of PMAC. Bit 3 of X:$0786 corresponds to A20 of the base address, and bit 0 of X:$0786 corresponds to A16. X:$078A is only used if 24-bit or 32-bit addressing has been selected with X:$0783 and X:$078C.

Bits A19 through A14 of the DPRAM VME base address must be set by the host computer after every power-on/reset by writing a byte over the bus to the "page select" register in the PMAC's VME mailbox IC at the mailbox base address + $0121. This must be done even with the single-page 8k x 16 standard

DPRAM option. With the extended DPRAM option, the host computer must write to the page select register every time a new page is accessed.

X:$078A is actually used at power-on/reset only, so to set or change bits 8 to 15 of the VME bus DPRAM base address, change the value of X:$078A, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$078A is copied at power-on/reset is X:$E00D bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M97) to change the active setup without a **SAVE** and reset.

X:$078B VME DPRAM Enable

Range: $00 - $FF

Units: None

Default: $60

X:$078B controls whether VME access to the DPRAM IC on the PMAC is enabled or not. It should be set to $60 if DPRAM is not present to disable access; it should be set to $E0 if DPRAM is present to enable access.

X:$078B is actually used at power-on/reset only, so to set or change the DPRAM enabling, change the value of X:$078B, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$078B is copied at power-on/reset is X:$E00E bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M98) to change the active setup without a **SAVE** and reset.

X:$078C VME Address Width Control

Range: $00 - $FF

Units: None

Default: $10

X:$078C controls the VME bus address width, with or without DPRAM. It should take one of six values in normal use:

- X:$078C = $00: 32-bit addressing, no DPRAM
- X:$078C = $10: 24-bit addressing, no DPRAM
- X:$078C = $30: 16-bit addressing, no DPRAM
- X:$078C = $80: 32-bit addressing, with DPRAM
- X:$078C = $90: 24-bit addressing, with DPRAM
- X:$078C = $B0: 16-bit addressing, with DPRAM

X:$078C is actually used at power-on/reset only, so to set or change the VME bus address width, change the value of X:$078C, store this new value to non-volatile flash memory with the **SAVE** command, and reset the card with the **$$$** command. The active register into which the value of X:$078C is copied at power-on/reset is X:$E00F bits 0 – 7. It is permissible to write to this register directly (suggested M-variable M99) to change the active setup without a **SAVE** and reset.

PMAC2 Servo IC Setup Bits and Registers

A few setup bits and registers in PMAC2 Servo ICs are not assigned I-variables, but still can be set and saved like I-variables.

X:$C005 etc. Bit 17 Encoder n Third-Channel Demux Control {PMAC2 only}

Range: 0 .. 1

Units: none

Default: 0

Note

In V1.17 and newer firmware, this bit has been incorporated into I9n5.

Bit 17 of a hardware channel's control word permits the "de-multiplexing" of the U, V, and W hall commutation sensor bits from the index-channel input for the encoder in the style of the Yaskawa incremental encoders. When bit 17 is set to 1, the U, V, W, and index ("Z") states are broken out of the third-channel ("C") input according to the four possible AB-quadrature states as shown in the following table:

A	B	C
1	1	Z
1	0	U
0	0	V
0	1	W

When bit 17 is set to 0, no de-multiplexing for the channel, and the U, V, W, and C input lines for the channel each feed their own status bits.

The addresses for the control word of each channel are:

Channel	Control Word Address	Channel	Control Word Address	Channel	Control Word Address	Channel	Control Word Address
1	X:$C005	3	X:$C015	5	X:$C025	7	X:$C035
2	X:$C00D	4	X:$C01D	6	X:$C02D	8	X:$C03D

The U, V, and W states can be read in bits 22, 21, and 20, respectively, of the channel's status word in the Servo ASIC. The index channel state can be read in bit 14 of the channel's status word; the state can be used automatically in the capture logic for the channel.

Bit 3 of the channel's status word is set to 1 until all four AB-quadrature states have been seen. The user should not attempt to use the de-multiplexed UVW state for brushless motor phasing until this bit has been cleared to 0.

X:$C005 etc. Bit 18 Encoder n Hardware 1/T Enable {PMAC2 only}

Range: 0 .. 1

Units: none

Default: 0

Note

In V1.17C and newer firmware, this bit can be accessed as I9n9.

Bit 18 of a hardware channel's control word permits the enabling of a special hardware 1/T sub-count estimation for the channel in the PMAC2 DSPGATE1 Servo ASIC. (This requires revision "D" or newer of the DSPGATE1 IC, which started shipping in 2002.)

If bit 18 is set to 1, the ASIC will automatically compute 12 bits of timer-based estimated sub-count data every SCLK encoder sample clock cycle (default 9.83 MHz). This sub-count position data is then available to enhance the resolution of the hardware capture and compare functions.

When bit 18 is set to the default value of 0, the hardware 1/T functionality is disabled.

The addresses for the control word of each channel are:

Channel	Control Word Address	Channel	Control Word Address	Channel	Control Word Address	Channel	Control Word Address
1	X:$C005	3	X:$C015	5	X:$C025	7	X:$C035
2	X:$C00D	4	X:$C01D	6	X:$C02D	8	X:$C03D

When the hardware 1/T functionality is enabled by setting bit 18 to 1, the registers needed for the traditional software 1/T in the encoder conversion table are no longer accessible. At these addresses (the first two Y-registers for the channel in the ASIC) are instead four 12-bit sub-registers for sub-count capture and compare data.

The hardware 1/T functionality is mainly intended for use with the ACC-51x high-resolution interpolator boards for sinusoidal encoders. That board produces 10 bits of fractional count resolution for servo feedback with its A/D converters, but that fractional data cannot be used for hardware capture or compare functions between servo cycles.

X:$C014, X:$C034 Servo IC m ADC Strobe Word {PMAC2 only}

Range: $000000 - $FFFFFF

Units: Serial Data Stream (MSB first, starting on rising edge of phase clock)

Default: $FFFFFE

X:$C014 and X:$C034 control the ADC strobe signal for Servo IC machine interface channels 1 – 4 and 5 – 8, respectively. The 24-bit word set by these registers is shifted out serially on the ADC_STROB lines, MSB first, one bit per ADC_CLK cycle starting on the rising edge of the phase clock.

In revisions "D" and newer of the DSPGATE1 Servo IC (beginning shipments in 2002), bit 0 (the LSB) of X:$C014 and X:$C034 is a control bit that determines whether the Servo IC will expect "header" information on the return data streams that precedes the numerical data from the ADCs. If bit 0 is 0, no header information is expected, and the low output from this bit is held until the next rising edge of the phase clock. This setting must be used on all earlier revisions of the DSPGATE1 Servo IC.

In revisions "D" and newer, if bit 0 of X:$C014 or X:$C034 is 1, up to 4 bits of header information can be accepted on the returned serial data streams from the ADCs (as with the ADCs in Delta Tau's Geo power block amplifiers). These bits are "rolled over" and end up in bits 0 – 3 of the ADC register in the Servo IC, and the numerical data ends up with its MSB in bit 23 of the ADC register. If fewer than 4 header bits are expected, the beginning of the strobe word should be delayed by setting the first bit(s) of the register to 0. Specifically, if (4 – *n*) header bits are expected, the first *n* bits of the register should be set to 0. In this setting, the ADC_STROB output is taken low and held low after bit 0 is shifted out.

The first bit that is a 1 creates a rising edge on the ADC_STROB output that is used typically as a start-convert signal. Some A/D converters just need this rising edge for the conversion; others need the signal to stay high all of the way through the conversion. Intermediate bits of the ADC_STROB output can be used to transmit other information in some applications.

The default value of $FFFFFE is suitable for use with Delta Tau Quad Amps, most third-party direct-PWM amplifiers, and with ACC-28B A/D converters. A value of $3FFFFF is usually appropriate for Delta Tau GEO power-block amplifiers. Refer to the specific amplifier manual for details. A valve of $1FFFFF should be used for the Option 12 serial ADCs on PMAC2a-PC/104.

PMAC I/O AND MEMORY MAP

This guide to PMAC's memory and input and output registers is provided for user reference. The PMAC architecture is very open, allowing the user to examine and use many internal registers for his own use. Usually this is done through the use of M-variables, which point to locations in the memory-I/O space of the PMAC processor. Once defined to point to the proper location, an M-variable can be treated as any other variable for reading and writing.

Caution:

Certain registers that are under PMAC's automatic control can cause problems if the user writes to them directly – particularly those used in the servo calculations.

PMAC's processor is the Motorola 56002 DSP. The 56002 has dual data buses, each 24 bits wide, so that both operands in a calculation may be brought in simultaneously. Each bus has access to a 16-bit address space (0000hex to FFFFhex) which provides 65,536 24-bit words. One bus and address space is called X, and the other is called Y. Therefore, when specifying a single-word memory location, one must use X: or Y: with the 16-bit address. PMAC's input and output is mapped into the same address space with the memory.

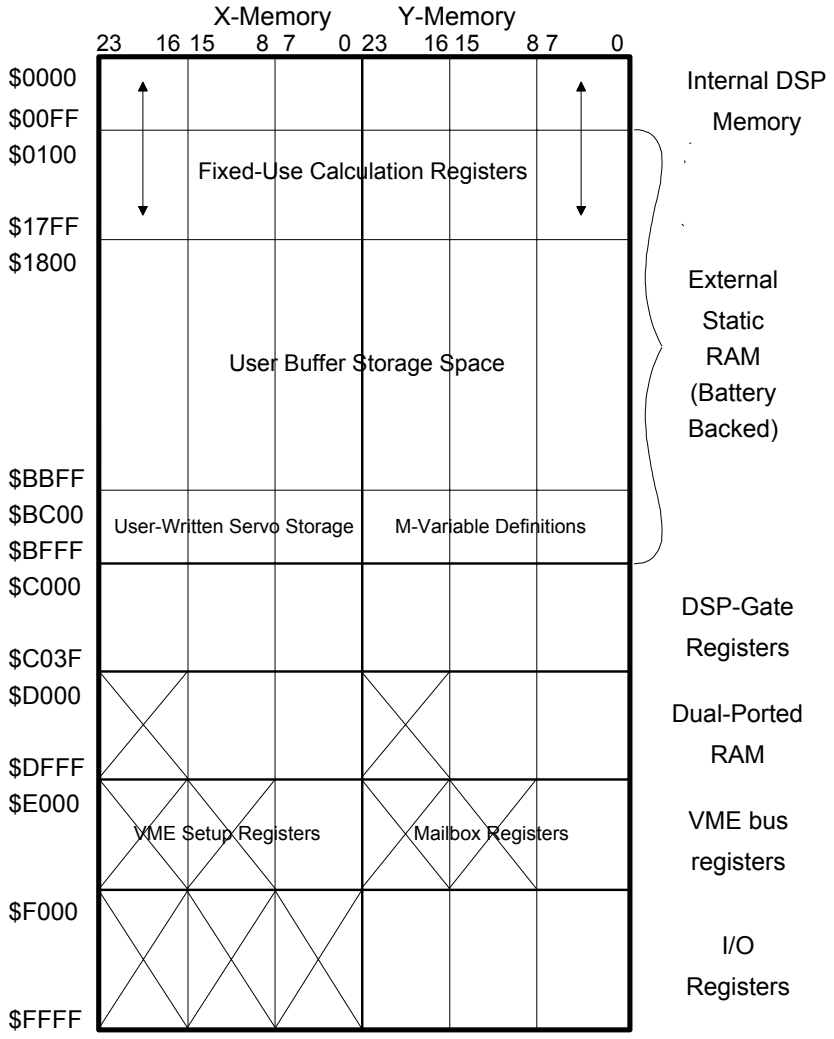
PMAC uses double-word memory for both extended fixed-point values and for floating-point values (single words are always fixed point). In this memory map, the fixed-point double word locations are specified by a D: (double), and the floating-point double-word locations are specified by an L: (long). This matches the syntax of M-variable declarations for these registers. If an address is specified without any prefix, it means that a special internal format is used to hold data in that word.

The user may specify PMAC addresses with either decimal or hexadecimal values; the hex values must be preceded by a `$` to be interpreted as hex. For example, `Y:$FFC0` is the hexadecimal specification, and `Y:65472` is the decimal specification of the same word address.

M-variables are defined by providing the word address, the offset, the width, and the format (irrelevant for bits). Refer to M-variables in the manual for details of the definition. Several M-variables were defined at the factory to match to inputs and outputs. For instance, M11 thru M18 were assigned to Machine Inputs 1 thru 8 (MI1-MI8), and M1 to M8 were assigned to Machine Outputs 1 thru 8 (MO1-MO8). Example statements are as follows:

```
M11->Y:$FFC2,0,1        (Mach. In. 1: Offset 0, width 1)
M1->Y:$FFC2,8,1         (Mach. Out. 1: Offset 8, width 1)
M162->D:$002C           (Motor #1 Actual Position: fixed pt.)
M163->L:$081F           (C.S. 1 X-axis Position: floating pt.)
```

PMAC Memory Mapping



Global Servo Calculation Registers

In the listing below, the hexadecimal address is listed first, followed by the decimal address in parentheses.

	Bits	Global I-Variables
X:$0000 (0)		Servo interrupt cycle counter (servo cycles)
Y:$0000		Servo count for next real time interrupt
X:$0001 (1)		
	0,1	I5
	2	I2
	4,5	I9
	6,7	I3
	10,11	I4
	14,15	I6
	18,19	I1

Y:$0001		Real time interrupt period minus one (I8)
X:$0002 (2)		Data gathering counter (time)
Y:$0002		Data gathering period (I19)
X:$0003 (3)		Global servo status bits
(First word returned on ??? command. See ??? in the On-Line Commands section)		
	0	This Card Addressed Serially
	1	All Cards Addressed Serially
	2-3	(Reserved for future use)
	4	MACRO Ring Failure Error
	5	MACRO Auxiliary Communications Error
	6	TWS Variable Parity Error
	7-9	(For internal use)
	10	EAROM Error
	11	DPRAM Error
	12	PROM Checksum Error
	13	Any Memory Checksum Error
	14	Compensation On
	15	PMAC2 Servo-Channel ADC Autocopy Disabled
	16	(Reserved for future use)
	17	Prepared to gather on trigger
	18	Prepared to gather next servo
	19	Data gathering function on
	20	Servo Error
	21	Servo Active
	22	Real Time Interrupt Re-entry
	23	Real Time Interrupt Active
Y:$0003		Global status bits
(Second word returned on ??? command)		
	0-7	(Reserved for future use)
	8-10	(Internal use)
	11	Fixed Buffer Full
	12-14	(Internal use)
	15	VME Communications Mode
	16	PLC Command
	17	PLC Buffer Open
	18	Rotary Buffer Open
	19	Motion Program Buffer Open
	20-21	(Internal use)
	22	Host communication mode
	23	(Internal use)
D:$0004-$0007		Temporary calculation registers
X:$0008 (8)		Last real-time interrupt execution time (servo cycles)
Y:$0008		Longest real-time interrupt execution time (servo cycles)
D:$0009-$001E		Temporary calculation registers
Y:$001F (31)		Minimum watchdog timer value, last background cycle

Motor Calculation Registers: PMAC(1), PID Servo Algorithm

This section provides addresses for motor calculation registers for PMAC(1) boards with the standard PID servo algorithm (without the Option 6 Extended Servo Algorithm).

The addresses given are for Motor #1. For the registers for another motor x, add (x-1)*$3C – (x-1)*60 – to the appropriate motor #1 address. The address table shown every 16 addresses (every 10hex) gives the matching addresses for motors 1-8 to make address calculations easier.

Example: Using the table, Motor 6's actual position address is $014C + ($28-$20) = $0154. Using the formula, it is (6-1)*$003C+$0028 = $0154.

Motor #	1	2	3	4	5	6	7	8
Hex	[$0020]	[$005C]	[$0098]	[$00D4]	[$0110]	[$014C]	[$0188]	[$01C4]
Decimal	32	92	152	212	272	332	392	452

D:$0020 (32)	Time left in move (X-register units msec*2 at %100)
D:$0021 (33)	Present desired jerk residual
D:$0022 (34)	Present desired jerk (dA/dt)
D:$0023 (35)	Present desired acceleration residual
D:$0024 (36)	Present desired acceleration (X-register units 6/[Ix08*32] cts/msec^2 at %100); Y is fractional
D:$0025 (37)	Present desired velocity residual
D:$0026 (38)	Present desired velocity (X-register units 3/[Ix08*32] cts/msec at %100); Y is fractional
D:$0027 (39)	Present desired position residual
D:$0028 (40)	Present desired position (1/[Ix08*32] counts)
X:$0029 (41)	Address of position feedback (Ix03)
Y:$0029	Position scaling factor (Ix08)
X:$002A (42)	Address of master (handwheel) register (Ix05)
Y:$002A	Previous actual position value
D:$002B (43)	Present actual position (1/[Ix08*32] counts)
X:$002C (44)	Master (handwheel) scale factor (Ix07)
Y:$002C	Previous master (handwheel) position
D:$002D (45)	Present master (handwheel) position (1/[Ix07*32]cts of the master or 1/[Ix08*32]cts of the slaved motor)
X:$002E (46)	Feedpot (timebase) pointer
Y:$002E	Servo cycle extension (Ix60)
D:$002F	Previous net desired position (1/[Ix08*32] counts)

Motor #	1	2	3	4	5	6	7	8
Hex	[$0030]	[$006C]	[$00A8]	[$00E4]	[$0120]	[$015C]	[$0198]	[$01D4]
Decimal	48	108	168	228	288	348	408	468

X:$0030 (48)	Velocity feedforward gain (Ix32)
Y:$0030	Previous desired velocity (1/[Ix08*32] cts/servo cycle)
X:$0031 (49)	Address of "velocity" encoder (Ix04)
Y:$0031	Previous "velocity" position
X:$0032 (50)	Acceleration feedforward gain (Ix35)
Y:$0032	"Velocity" scaling factor (Ix09)
X:$0033 (51)	Actual velocity (1/[Ix09*32] cts/[Ix60+1]servo interrupts)
Y:$0033	Derivative gain (Ix31)

X:$0034 (52)	Deadband size (Ix65) (1/16 count)
Y:$0034	Position error limit (Ix67) (1/16 count)
X:$0035 (53)	"Deadband gain" (Ix64)
Y:$0035	Integral gain (Ix33)
X:$0036 (54)	Integrated error residual
Y:$0036	Integrated error limit (Ix63)
D:$0037 (55)	Integrated error
X:$0038 (56)	Proportional gain (Ix30)
Y:$0038	Filter output (DAC) limit (Ix69)
D:$0039	(Filter intermediate values)
X:$003A (58)	Filter result (stored for next cycle)
X:$003B (59)	Notch filter D2 gain (Ix39)
Y:$003B	Notch filter N2 gain (Ix37)
X:$003C (60)	Notch filter D1 gain (Ix38)
Y:$003C	Notch filter N1 gain (Ix36)
X:$003D (61)	Motor servo status bits

Bits

(First word returned on **?** command. See `Y:$0814` for second word.)
(Refer to ? description in on-line commands for detailed description of bits.)

	0-9	(Internal use)
	10	Home search in progress
	11	Block request
	12	Abort deceleration in progress
	13	Desired velocity 0
	14	Data block error
	15	Dwell in progress
	16	Integration mode (Ix34; 0 on always; 1 on when desired velocity zero)
	17	Move timer active
	18	Open-loop mode
	19	Phased motor (Ix01)
	20	Handwheel enabled (Ix06)
	21	Positive end limit set (soft or hard [**-LIM**])
	22	Negative end limit set (soft or hard [**+LIM**])
	23	Motor activated (Ix00)
Y:$003D		Phase address pointer (Ix83)
X:$003E (62)	0-7	Phase offset (Ix72)
	8-23	2nd phase bias (Ix79)
Y:$003E	0-7	# of commutation cycles per rev (Ix70)
	8-23	Filter output/1st phase bias (Ix29)
X:$003F (63)		Magnetization current (Ix77)
Y:$003F		Previous phase position

Motor #	1	2	3	4	5	6	7	8
Hex	**[$0040]**	**[$007C]**	**[$00B8]**	**[$00F4]**	**[$0130]**	**[$016C]**	**[$01A8]**	**[$01E4]**
Decimal	64	124	184	244	304	364	424	484

X:$0040 (64)	Slip frequency
Y:$0040 (64)	Counts per Ix70 commutation cycles (Ix71)
D:$0041 (65)	Present phase position (X register units: counts*Ix70, range -Ix71/2 - Ix71/2-1)

X:$0042 (66)	Phase advance	
Y:$0042	Phase advance gain (Ix76)	
X:$0043 (67)	Slip Gain (Ix78)	
Y:$0043	Phased DAC amplitude	
X:$0044 (68)	Command output address (Ix02)	
Y:$0044	Velocity Phase Advance Gain (Ix76)	
X:$0045 (69)	Filter command value	
Y:$0045	Command (torque) internal offset	
D:$0046 (70)	Compensation correction (1/[Ix08*32] cts	
D:$0047 (71)	Following error (1/[Ix08*32] cts)	
X:$0048 (72)	Friction feedforward gain (Ix68)	
$0049-$005B	(Reserved for future use)	
$005C-$0097	Motor #2 registers (as above)	(92-151)
$0098-$00D3	Motor #3 registers (as above)	(152-211)
$00D4-$010F	Motor #4 registers (as above)	(212-271)
$0110-$014B	Motor #5 registers (as above)	(272-331)
$014C-$0187	Motor #6 registers (as above)	(332-391)
$0188-$01C3	Motor #7 registers (as above)	(392-451)
$01C4-$01FF	Motor #8 registers (as above)	(452-511)

Motor Calculation Registers: PMAC(1), Extended Servo Algorithm (ESA)

This section provides addresses for motor calculation registers for PMAC(1) boards with the Option 6 Extended Servo Algorithm.

(The addresses given are for Motor #1. For the registers for another motor x, add (x-1)*$3C – (x-1)*60 – to the appropriate motor #1 address. The address table shown every 16 addresses (every 10hex) gives the matching addresses for motors 1-8 to make address calculations easier.

Example: Using the table, Motor 6 actual position address is $014C + ($28-$20) = $0154. Using the formula, it is (6-1)*$003C+$0028 = $0154

Motor #	1	2	3	4	5	6	7	8
Hex	**[$0020]**	**[$005C]**	**[$0098]**	**[$00D4]**	**[$0110]**	**[$014C]**	**[$0188]**	**[$01C4]**
Decimal	32	92	152	212	272	332	392	452

D:$0020 (32)	Time left in move (X-register units msec*2 at %100)
D:$0021 (33)	Present desired jerk residual
D:$0022 (34)	Present desired jerk (dA/dt)
D:$0023 (35)	Present desired acceleration residual
D:$0024 (36)	Present desired acceleration (X-register units 6/[Ix08*32] cts/msec^2 at %100); Y is fractional
D:$0025 (37)	Present desired velocity residual
D:$0026 (38)	Present desired velocity (X-register units 3/[Ix08*32] cts/msec at %100); Y is fractional
D:$0027 (39)	Present desired position residual
D:$0028 (40)	Present desired position (1/[Ix08*32] counts)
X:$0029 (41)	Address of position feedback (Ix03)
Y:$0029	Position scaling factor (Ix08)
X:$002A (42)	Address of master (handwheel) register (Ix05)
Y:$002A	Previous actual position value

D:$002B (43) Present actual position (1/[Ix08*32] counts)
X:$002C (44) Master (handwheel) scale factor (Ix07)
Y:$002C Previous master (handwheel) position
D:$002D (45) Present master (handwheel) position (1/[Ix07*32]cts of the master or 1/[Ix08*32]cts of the slaved motor)
X:$002E (46) Feedpot (timebase) pointer
Y:$002E (46) Servo extension counter
X:$002F (47) ESA S0 gain (Ix30)
Y:$002F (47) Servo cycle extension (Ix60)

Motor #	1	2	3	4	5	6	7	8
Hex	**[$0030]**	**[$006C]**	**[$00A8]**	**[$00E4]**	**[$0120]**	**[$015C]**	**[$0198]**	**[$01D4]**
Decimal	48	108	168	228	288	348	408	468

X:$002E (46) Feedpot (timebase) pointer
D:$0030 (48) Compensation correction (1/[Ix08*32] cts)
D:$0031 (48) Following error (1/[Ix08*32] cts)
X:$0032 (50) ESA T0 gain (Ix40)
Y:$0032 ESA S1 gain (Ix31)
X:$0033 (51) ESA UT1 term
Y:$0033 ESA T1 gain (Ix41)
X:$0034 (52) ESA UT2 term
Y:$0034 ESA T2 gain (Ix42)
X:$0035 (53) ESA UT3 term
Y:$0035 ESA T3 gain (Ix43)
X:$0036 (54) ESA UT4 term
Y:$0036 ESA T4 gain (Ix44)
X:$0037 (55) ESA UR1 term
Y:$0037 ESA R1 gain (Ix36)
X:$0038 (56) ESA UR2 term
Y:$0038 ESA R2 gain (Ix37)
X:$0039 (57) ESA UR3 term
Y:$0039 ESA R3 gain (Ix38)
X:$003A (58) ESA UR4 term
Y:$003A ESA R4 gain (Ix39)
X:$003B (59) Address of "velocity" encoder (Ix04)
Y:$003B ESA TS gain (Ix45)
X:$003C (60) ESA F0 gain (Ix32)
Y:$003C Velocity scaling factor (Ix09)
X:$003D (61) Actual velocity (1/[Ix09*32] cts/[Ix60+1]servo interrupts)
Y:$003D Previous velocity-loop source position
D:$003E (62) Previous velocity-loop extended actual position
X:$003F (63) ESA F1 gain (Ix33)
Y:$003F ESA G0 gain (Ix56)

Motor #	1	2	3	4	5	6	7	8
Hex	**[$0040]**	**[$007C]**	**[$00B8]**	**[$00F4]**	**[$0130]**	**[$016C]**	**[$01A8]**	**[$01E4]**
Decimal	64	124	184	244	304	364	424	484

X:$0040 (64) ESA UG1 term
Y:$0040 ESA G1 gain (Ix57)
X:$0041 (65) ESA UG2 term
Y:$0041 ESA G2 gain (Ix58)
X:$0042 (66) ESA UD1 term
Y:$0042 ESA D1 gain (Ix54)
X:$0043 (67) ESA UD2 term
Y:$0043 ESA D2 gain (Ix55)
X:$0044 (68) ESA GS gain (Ix58)
Y:$0044 ESA H0 gain (Ix34)
D:$0045 (69) Previous net desired position
X:$0046 (70) ESA H1 gain (Ix35)
Y:$0046 ESA K0 gain (Ix49)
X:$0047 (71) ESA UK1 term
Y:$0047 ESA K1 gain (Ix50)
X:$0048 (72) ESA UK2 term
Y:$0048 ESA K2 gain (Ix51)
X:$0049 (73) ESA UK3 term
Y:$0049 ESA K3 gain (Ix52)
X:$004A (74) ESA UL1 term
Y:$004A ESA L1 gain (Ix46)
X:$004B (75) ESA UL2 term
Y:$004B ESA L2 gain (Ix47)
X:$004C (76) ESA UL3 term
Y:$004C ESA L3 gain (Ix48)
X:$004D (77) ESA KS gain (Ix53)
Y:$004D (Filter intermediate value)
X:$004E (78) Previous filter (DAC) output
Y:$004E Filter output (DAC) limit (Ix69)
X:$004F (79) Motor servo status bits

(First word returned on **?** command. See **Y:$0814** for second word.)
(Refer to ? description in on-line commands for detailed description of bits.)

0-9	(Internal use)
10	Home search in progress
11	Block request
12	Abort deceleration in progress
13	Desired velocity 0
14	Data block error
15	Dwell in progress
16	(Reserved for future use)
17	Move timer active
18	Open-loop mode
19	Phased motor (Ix01)
20	Handwheel enabled (Ix06)
21	Positive end limit set (soft or hard [**-LIM**])
22	Negative end limit set (soft or hard [**+LIM**])
23	Motor activated (Ix00)

Y:$004F Phase address pointer (Ix83)

Motor #	1	2	3	4	5	6	7	8
Hex	[$0050]	[$008C]	[$00C8]	[$0104]	[$0140]	[$017C]	[$01B8]	[$01F4]
Decimal	80	140	200	260	320	380	440	500

X:$0050 (80)

0-7	Phase offset (Ix72)
8-23	2nd phase bias (Ix79)

Y:$0050

0-7	# of commutation cycles per rev (Ix70)
8-23	Filter output/1st phase bias (Ix29)

X:$0051 (81) Magnetization current (Ix77)
Y:$0051 Previous phase position
X:$0052 (82) Slip frequency
Y:$0052 Counts per Ix70 commutation cycles (Ix71)
D:$0053 (83) Present phase position (X register units: counts*Ix70, range -Ix71/2 - Ix71/2-1)
X:$0054 (84) Phase advance
Y:$0054 Phase advance gain (Ix76)
X:$0055 (85) Slip Gain (Ix78)
Y:$0055 Phased DAC amplitude
X:$0056 (86) Command output address (Ix02)
Y:$0056 Velocity Phase Advance Gain (Ix76)
X:$0057 (87) Servo command value
Y:$0057 Command internal offset
X:$0058 (88) Velocity node internal offset
$0059-$005B (Reserved for future use)

$005C-$0097	Motor #2 registers (as above)	(92-151)
$0098-$00D3	Motor #3 registers (as above)	(152-211)
$00D4-$010F	Motor #4 registers (as above)	(212-271)
$0110-$014B	Motor #5 registers (as above)	(272-331)
$014C-$0187	Motor #6 registers (as above)	(332-391)
$0188-$01C3	Motor #7 registers (as above)	(392-451)
$01C4-$01FF	Motor #8 registers (as above)	(452-511)

Motor Calculation Registers: PMAC2, PID Servo Algorithm

This section provides addresses for motor calculation registers for PMAC2 boards with the standard PID servo algorithm (without the Option 6 Extended Servo Algorithm).

(The addresses given are for Motor #1. For the registers for another motor x, add (x-1)*$3C – (x-1)*60 – to the appropriate motor #1 address.) The address table shown every 16 addresses (every 10hex) gives the matching addresses for motors 1-8 to make address calculations easier.

Example: Using the table, Motor 6 actual position address is $014C + ($28-$20) = $0154. Using the formula, it is (6-1)*$003C+$0028 = $0154.

Motor #	1	2	3	4	5	6	7	8
Hex	[$0020]	[$005C]	[$0098]	[$00D4]	[$0110]	[$014C]	[$0188]	[$01C4]
Decimal	32	92	152	212	272	332	392	452

D:$0020 (32) Time left in move (X-register units msec*2 at %100)
D:$0021 (33) Present desired jerk residual
D:$0022 (34) Present desired jerk (dA/dt)

D:$0023 (35)	Present desired acceleration residual
D:$0024 (36)	Present desired acceleration (X-register units 6/[Ix08*32] cts/msec^2 at %100); Y is fractional
D:$0025 (37)	Present desired velocity residual
D:$0026 (38)	Present desired velocity (X-register units 3/[Ix08*32] cts/msec at %100); Y is fractional
D:$0027 (39)	Present desired position residual
D:$0028 (40)	Present desired position (1/[Ix08*32] counts)
X:$0029 (41)	Address of position feedback (Ix03)
Y:$0029	Position scaling factor (Ix08)
X:$002A (42)	Address of master (handwheel) register (Ix05)
Y:$002A	Previous actual position value
D:$002B (43)	Present actual position (1/[Ix08*32] counts)
X:$002C (44)	Master (handwheel) scale factor (Ix07)
Y:$002C	Previous master (handwheel) position
D:$002D (45)	Present master (handwheel) position (1/[Ix07*32]cts of the master or 1/[Ix08*32]cts of the slaved motor)
X:$002E (46)	Feedpot (timebase) pointer
Y:$002E	Servo cycle extension (Ix60)
D:$002F	Previous net desired position (1/[Ix08*32] counts)

Motor #	1	2	3	4	5	6	7	8
Hex	**[$0030]**	**[$006C]**	**[$00A8]**	**[$00E4]**	**[$0120]**	**[$015C]**	**[$0198]**	**[$01D4]**
Decimal	48	108	168	228	288	348	408	468

X:$0030 (48)	Velocity feedforward gain (Ix32)
Y:$0030	Previous desired velocity
X:$0031 (49)	Address of "velocity" encoder (Ix04)
Y:$0031	Previous "velocity" position
X:$0032 (50)	Acceleration feedforward gain (Ix35)
Y:$0032	"Velocity" scaling factor (Ix09)
X:$0033 (51)	Actual velocity (1/[Ix09*32] cts/[Ix60+1]servo interrupts)
Y:$0033	Derivative gain (Ix31)
X:$0034 (52)	Deadband size (Ix65) (1/16 count)
Y:$0034	Position error limit (Ix67) (1/16 count)
X:$0035 (53)	"Deadband gain" (Ix64)
Y:$0035	Integral gain (Ix33)
X:$0036 (54)	Integrated error residual
Y:$0036	Integrated error limit (Ix63)
D:$0037 (55)	Integrated error
X:$0038 (56)	Proportional gain (Ix30)
Y:$0038	Filter output (DAC) limit (Ix69)
D:$0039	(Filter intermediate values)
X:$003A (58)	Filter result (stored for next cycle)
X:$003B (59)	Notch filter D2 gain (Ix39)
Y:$003B	Notch filter N2 gain (Ix37)
X:$003C (60)	Notch filter D1 gain (Ix38)
Y:$003C	Notch filter N1 gain (Ix36)
X:$003D (61)	Motor servo status bits

(First word returned on ? command. See `Y:$0814` for second word.)
(Refer to ? description in on-line commands for detailed description of bits)

	0-9	(Internal use)
	10	Home search in progress
	11	Block request
	12	Abort deceleration in progress
	13	Desired velocity 0
	14	Data block error
	15	Dwell in progress
	16	Integration mode (Ix34; 0 on always; 1 on when desired velocity zero)
	17	Move timer active
	18	Open-loop mode
	19	Phased motor (Ix01)
	20	Handwheel enabled (Ix06)
	21	Positive end limit set (soft or hard [**PLIM**])
	22	Negative end limit set (soft or hard [**MLIM**])
	23	Motor activated (Ix00)
Y:$003D		Phase address pointer (Ix83)
X:$003E (62)		
	0-7	Phase offset (Ix72)
	8-23	2nd phase bias (Ix79)
Y:$003E		
	0-7	# of commutation cycles per rev (Ix70)
	8-23	Filter output/1st phase bias (Ix29)
X:$003F		Counts per Ix70 commutation cycles (Ix71)
Y:$003F		Previous phase position

Motor #	1	2	3	4	5	6	7	8
Hex	**[$0040]**	**[$007C]**	**[$00B8]**	**[$00F4]**	**[$0130]**	**[$016C]**	**[$01A8]**	**[$01E4]**
Decimal	64	124	184	244	304	364	424	484

D:$0040 (64)	Present phase position (X-register units: counts*Ix70)
X:$0041 (65)	ADC mask word (Ix84)
Y:$0041	ADC address (Ix82)
X:$0042 (66)	Estimated rotor magnetization current
Y:$0042	Slip gain (Ix78)
X:$0043 (67)	Commanded quadrature current (servo command)
Y:$0043	Commanded direct current (Ix77)
X:$0044 (68)	Actual quadrature current
Y:$0044	Actual direct current
X:$0045 (69)	Current-loop back path proportional gain (Ix76)
Y:$0045	Current-loop forward path proportional gain (Ix62)
X:$0046 (70)	Quadrature current loop integrator output
Y:$0046	Direct current loop integrator output
X:$0047 (71)	PWM scale factor (Ix66)
Y:$0047	Current-loop integral gain (Ix61)
X:$0048 (72)	Command output address (Ix02)
Y:$0048	Friction feedforward gain (Ix68)
X:$0049 (73)	(Quadrature) command value
Y:$0049	(Quadrature) command bias

D:$004A (74)	Compensation correction (1/[Ix08*32] cts)	
D:$004B (75)	Following error (1/[Ix08*32] cts)	
$004C-$005B	(Reserved for future use)	
$005C-$0097	Motor #2 registers (as above)	(92-151)
$0098-$00D3	Motor #3 registers (as above)	(152-211)
$00D4-$010F	Motor #4 registers (as above)	(212-271)
$0110-$014B	Motor #5 registers (as above)	(272-331)
$014C-$0187	Motor #6 registers (as above)	(332-391)
$0188-$01C3	Motor #7 registers (as above)	(392-451)
$01C4-$01FF	Motor #8 registers (as above)	(452-511)

Motor Calculation Registers: PMAC2, Extended Servo Algorithm (ESA)

This section provides addresses for motor calculation registers for PMAC2 boards with the Option 6 Extended Servo Algorithm.

(The addresses given are for Motor #1. For the registers for another motor x, add (x-1)*$3C – (x-1)*60 – to the appropriate motor #1 address.) The address table shown every 16 addresses (every 10hex) gives the matching addresses for motors 1-8 to make address calculations easier.

Example: Using the table, Motor 6 actual position address is $014C + ($28-$20) = $0154. Using the formula, it is (6-1)*$003C+$0028 = $0154.

Motor #	1	2	3	4	5	6	7	8
Hex	**[$0020]**	**[$005C]**	**[$0098]**	**[$00D4]**	**[$0110]**	**[$014C]**	**[$0188]**	**[$01C4]**
Decimal	32	92	152	212	272	332	392	452

D:$0020 (32)	Time left in move (X-register units msec*2 at %100)
D:$0021 (33)	Present desired jerk residual
D:$0022 (34)	Present desired jerk (dA/dt)
D:$0023 (35)	Present desired acceleration residual
D:$0024 (36)	Present desired acceleration (X-register units 6/[Ix08*32] cts/msec^2 at %100); Y is fractional
D:$0025 (37)	Present desired velocity residual
D:$0026 (38)	Present desired velocity (X-register units 3/[Ix08*32] cts/msec at %100); Y is fractional
D:$0027 (39)	Present desired position residual
D:$0028 (40)	Present desired position (1/[Ix08*32] counts)
X:$0029 (41)	Address of position feedback (Ix03)
Y:$0029	Position scaling factor (Ix08)
X:$002A (42)	Address of master (handwheel) register (Ix05)
Y:$002A	Previous actual position value
D:$002B (43)	Present actual position (1/[Ix08*32] counts)
X:$002C (44)	Master (handwheel) scale factor (Ix07)
Y:$002C	Previous master (handwheel) position
D:$002D (45)	Present master (handwheel) position (1/[Ix07*32]cts of the master or 1/[Ix08*32]cts of the slaved motor).
X:$002E (46)	Feedpot (timebase) pointer
Y:$002E (46)	Servo extension counter
X:$002F (47)	ESA S0 gain (Ix30)
Y:$002F (47)	Servo cycle extension (Ix60)

Motor #	1	2	3	4	5	6	7	8
Hex	[$0030]	[$006C]	[$00A8]	[$00E4]	[$0120]	[$015C]	[$0198]	[$01D4]
Decimal	48	108	168	228	288	348	408	468

D:$0030 (48) Compensation correction (1/[Ix08*32] cts)
D:$0031 (48) Following error (1/[Ix08*32] cts)
X:$0032 (50) ESA T0 gain (Ix40)
Y:$0032 ESA S1 gain (Ix31)
X:$0033 (51) ESA UT1 term
Y:$0033 ESA T1 gain (Ix41)
X:$0034 (52) ESA UT2 term
Y:$0034 ESA T2 gain (Ix42)
X:$0035 (53) ESA UT3 term
Y:$0035 ESA T3 gain (Ix43)
X:$0036 (54) ESA UT4 term
Y:$0036 ESA T4 gain (Ix44)
X:$0037 (55) ESA UR1 term
Y:$0037 ESA R1 gain (Ix36)
X:$0038 (56) ESA UR2 term
Y:$0038 ESA R2 gain (Ix37)
X:$0039 (57) ESA UR3 term
Y:$0039 ESA R3 gain (Ix38)
X:$003A (58) ESA UR4 term
Y:$003A ESA R4 gain (Ix39)
X:$003B (59) Address of "velocity" encoder (Ix04)
Y:$003B ESA TS gain (Ix45)
X:$003C (60) ESA F0 gain (Ix32)
Y:$003C Velocity scaling factor (Ix09)
X:$003D (61) Actual velocity (1/[Ix09*32] cts/[Ix60+1]servo interrupts)
Y:$003D Previous velocity-loop source position
D:$003E (62) Previous velocity-loop extended actual position
X:$003F (63) ESA F1 gain (Ix33)
Y:$003F ESA G0 gain (Ix56)

Motor #	1	2	3	4	5	6	7	8
Hex	[$0040]	[$007C]	[$00B8]	[$00F4]	[$0130]	[$016C]	[$01A8]	[$01E4]
Decimal	64	124	184	244	304	364	424	484

X:$0040 (64) ESA UG1 term
Y:$0040 ESA G1 gain (Ix57)
X:$0041 (65) ESA UG2 term
Y:$0041 ESA G2 gain (Ix58)
X:$0042 (66) ESA UD1 term
Y:$0042 ESA D1 gain (Ix54)
X:$0043 (67) ESA UD2 term
Y:$0043 ESA D2 gain (Ix55)
X:$0044 (68) ESA GS gain (Ix58)
Y:$0044 ESA H0 gain (Ix34)
D:$0045 (69) Previous net desired position
X:$0046 (70) ESA H1 gain (Ix35)

Y:$0046 ESA K0 gain (Ix49)
X:$0047 (71) ESA UK1 term
Y:$0047 ESA K1 gain (Ix50)
X:$0048 (72) ESA UK2 term
Y:$0048 ESA K2 gain (Ix51)
X:$0049 (73) ESA UK3 term
Y:$0049 ESA K3 gain (Ix52)
X:$004A (74) ESA UL1 term
Y:$004A ESA L1 gain (Ix46)
X:$004B (75) ESA UL2 term
Y:$004B ESA L2 gain (Ix47)
X:$004C (76) ESA UL3 term
Y:$004C ESA L3 gain (Ix48)
X:$004D (77) ESA KS gain (Ix53)
Y:$004D (Filter intermediate value)
X:$004E (78) Previous filter (DAC) output
Y:$004E Filter output (DAC) limit (Ix69)
X:$004F (79) Motor servo status bits
(First word returned on **?** command. See `Y:$0814` for second word.)
(Refer to ? description in on-line commands for detailed description of bits)

0-9	(Internal use)
10	Home search in progress
11	Block request
12	Abort deceleration in progress
13	Desired velocity 0
14	Data block error
15	Dwell in progress
16	(Reserved for future use)
17	Move timer active
18	Open-loop mode
19	Phased motor (Ix01)
20	Handwheel enabled (Ix06)
21	Positive end limit set (soft or hard [**PLIM**])
22	Negative end limit set (soft or hard [**MLIM**])
23	Motor activated (Ix00)

Y:$004F Phase address pointer (Ix83)

Motor #	1	2	3	4	5	6	7	8
Hex	**[$0050]**	**[$008C]**	**[$00C8]**	**[$0104]**	**[$0140]**	**[$017C]**	**[$01B8]**	**[$01F4]**
Decimal	80	140	200	260	320	380	440	500

X:$0050 (80) 0-7 Phase offset (Ix72)
8-23 2nd phase bias (Ix79)
Y:$0050 0-7 # of commutation cycles per rev (Ix70)
8-23 Filter output/1st phase bias (Ix29)
X:$0051 (81) Counts per Ix70 commutation cycles (Ix71)
Y:$0051 Previous phase position
D:$0052 (82) Present phase position (X-register units: counts*Ix70)
X:$0053 (83) ADC mask word (Ix84)
Y:$0053 ADC address (Ix82)
X:$0054 (84) Estimated rotor magnetization current

Y:$0054	Slip gain (Ix78)	
X:$0055 (85)	Commanded quadrature current (servo command)	
Y:$0055	Commanded direct current (Ix77)	
X:$0056 (86)	Actual quadrature current	
Y:$0056	Actual direct current	
X:$0057 (87)	Current-loop back path proportional gain (Ix76)	
Y:$0057	Current-loop forward path proportional gain (Ix62)	
X:$0058 (88)	Quadrature current loop integrator output	
Y:$0058	Direct current loop integrator output	
X:$0059 (89)	PWM scale factor (Ix66)	
Y:$0059	Current-loop integral gain (Ix61)	
X:$005A (90)	Command output address (Ix02)	
Y:$005A	Friction feedforward gain (Ix68)	
X:$005B (91)	(Quadrature) command value	
Y:$005B	(Quadrature) command bias	
$005C-$0097	Motor #2 registers (as above)	(92-151)
$0098-$00D3	Motor #3 registers (as above)	(152-211)
$00D4-$010F	Motor #4 registers (as above)	(212-271)
$0110-$014B	Motor #5 registers (as above)	(272-331)
$014C-$0187	Motor #6 registers (as above)	(332-391)
$0188-$01C3	Motor #7 registers (as above)	(392-451)
$01C4-$01FF	Motor #8 registers (as above)	(452-511)

Buffers

X:$0400-$04FF	Commutation sine table [2^{23}*SIN((address-$400)*360°/256)]	(1024-1279)
X:$0600-$06FF	Command character queue	(512-767)
Y:$0600-$06FF	Response character queue	
X:$0700-$0701	User count-down timer registers (servo cycles)	(1792-1793)
Y:$0700-$0701	User count-down timer registers (servo cycles)	
X:$0708-$070F	Auto-converted ACC-36 ADC registers 9-16	(1800-1807)
Y:$0708-$070F	Auto-converted ACC-36 ADC registers 1-8	

Encoder Conversion (Interpolation) Table

X:$0720-$073F	Converted encoder and time base data	(1824-1855)
Y:$0720-$073F	Encoder conversion source and format	

The format of the conversion table is:

Y:word	**Bits**	**Conversion Format:**
	16-23	$00 = 1/T interpolation of incremental encoder
		$10 = A/D register conversion
		$20 = Unfiltered parallel Y word source*
		$30 = Filtered parallel Y word source**
		$40 = Time base*
		$50 = Integrated A/D register conversion
		$60 = Unfiltered parallel X word source*
		$70 = Filtered parallel X word source**
		$80 = Parallel interp. of incremental
		$90 = Triggered time base; frozen*
		$A0 = Triggered time base; running*
		$B0 = Triggered time base; armed*
		$C0 = no interpolation of incremental encoder
		$D0 = Exponential filter**
		$F0 = High-res. Interpolation of sinusoidal encoder
	0-15	Address of source data

* Next Y word contains user-set constant for conversion (this is a double-entry conversion).
** Next two Y words contain user-set constants for conversion (this is a triple-entry conversion).

X:word	0-4	Fractional bits of converted data
	5-23	Integer bits of converted data (if last entry in conversion) Intermediate value if not last entry in conversion

Refer to the detailed description of the encoder conversion table under “Feedback Features”.

General Global Registers

$0770 - $077F	Open memory; cleared to 0 on power-on/reset (useful for 24-bit M-variables)	(1904-1919)
Y:$0780 - Y:$07D1	LCD Display character memory	(1920-2001)
X:$0780 (1920)	Dwell (fixed) feedpot (I10)	
X:$0781 (1921)	Jog-to-position move delay time (I12)	
X:$0782 (1922)	Programmed move delay time (I11)	
X:$0783 - X:$078C	VME address and vector values (See VME Interface Document for details)	(1923 - 1932)
X:$0786 - X:$0787	PC Dual-ported RAM host address registers (See Option 2 Manual for details)	(1926 - 1927)
X:$078D (1933)	Move segmentation time (I13)	
X:$078E (1934)	Hold feedpot value (must be zero)	
X:$078F (1935)	Addressed motor/coordinate system word	

Bits

0-2	Motor number minus 1 (#n-1)
3-5	Coordinate system number minus 1 (&n-1)

X:$0790 (1936)	Card software address
X:$0794 (1940)	Running software checksum value (frozen if any error)
X:$07A1 (1953)	# of MACRO ring errors since power-on/reset
X:$07B1 (1969)	PROM (Firmware) Reference checksum value
X:$07B2 (1970)	User Program Reference checksum value
X:$07B3 (1971)	Defined buffer checksum value
X:$07B9 (1977)	Data gathering source mask (I20)
X:$07BA - X:$07D1 (1978 - 2001)	Data gathering source addresses (I21-I44)
Y:$07EF (2031)	Watchdog timer minimum value since power-on/reset
$07F0 - $07FF (2032 - 2047)	Open registers (useful for creating 24-bit M-variables to handle rollover of other 24-bit registers); on PMACs without flash memory (no option CPU) these are stored in battery-backed RAM

Motor and Coordinate System Status and Control Registers

The addresses given in this section are for Motor 1 and Coordinate System 1. Table headers for each set of 16 registers give the starting addresses of the set for each Motor/C.S. For the register of another Motor x or C.S. x, add (x-1)*$C0 or (x-1)*192 to the appropriate Motor 1 or C.S. 1 address.

Example: For the Motor 4 target position register, using the table compute $0A40 + ($080B-$0800) = $0A4B. Using the formula, compute (4-1)*$C0 + $080B = $0A4B

Mot/CS	1	2	3	4	5	6	7	8
Hex	**[$0800]**	**[$08C0]**	**[$0980]**	**[$0A40]**	**[$0B00]**	**[$0BC0]**	**[$0C80]**	**[$0D40]**
Decimal	2048	2240	2432	2624	2816	3008	3200	3392

L:$0800 (2048)	Motor #1 jog speed (Ix22) [floating point]
L:$0801 (2049)	Motor #1 jog max. accel (Ix19) [float. pt.]
X:$0802 (2050)	Motor #1 jog/home S-curve time (Ix21)
Y:$0802	Motor #1 jog/home accel time (Ix20)
X:$0803 (2051)	Motor #1 flag pointer and mode (Ix25)
Y:$0803	Motor #1 fatal following error limit (Ix11)
L:$0804 (2052)	Motor #1 limit/abort accel (Ix15) [flt. pt.]
X:$0805 (2053)	Motor #1 in-position band (Ix28)

Y:$0805 Motor #1 soft following error (Ix12)
X:$0806 (2054) C.S. 1 present timebase (units of I10 {1/8,388,608 msec})
X:$0807 (2055) C.S. 1 desired timebase pointer (Ix93)
Y:$0807 C.S. 1 present timebase pointer
X:$0808 (2056) C.S. 1 host command timebase [100% when = I10]
Y:$0808 C.S. 1 present timebase slew rate
X:$0809 (2057) C.S. 1 feed-hold timebase slew rate (Ix95)
Y:$0809 C.S. 1 timebase slew rate (Ix94)
D:$080A (2058) Motor #1 target velocity
D:$080B (2059) Motor #1 target position (1/[Ix08*32] cts)

Mot/CS	1	2	3	4	5	6	7	8
Hex	[$0810]	[$08D0]	[$0990]	[$0A50]	[$0B10]	[$0BD0]	[$0C90]	[$0D50]
Decimal	2064	2256	2448	2640	2832	3024	3216	3408

X:$0810 (2064) C.S. 1 move S-curve time
Y:$0810 C.S. 1 move accel time
L:$0811 (2065) C.S. 1 move feedrate or time [flt. pt.]
L:$0812 (2066) C.S. 1 feedrate time units (Ix90) [flt. pt.]
D:$0813 (2067) Motor #1 position bias (1/[Ix08*32] cts)
X:$0814 (2068) Motor #1 home offset position (Ix26)
Y:$0814 Motor #1 status bits
(Second word returned on **?** command. See X:$003D for first word.)
(Refer to **?** specification in On-line Commands for detailed description of bit meanings.)

Bits

0	In-position true
1	Warning following error limit exceeded
2	Fatal following error limit exceeded
3	Amplifier fault error
4	Backlash direction flag
5	I^2T Amplifier fault error
6	Integrated fatal following error
7	Trigger move
8	Phasing search error
9	(Reserved for future use)
10	Home complete
11	Stopped on position limit
12-13	(Reserved for future use)
14	Amplifier enabled
15-19	(Reserved for future use)
20-22	Number of C.S. defined in (-1)
23	Assigned to C.S.

X:$0815 (2069) Motor #1 phase finding torque and time (Ix73, Ix74)
Y:$0815 Motor #1 encoder home position offset (cts)
X:$0816 (2070) C.S. 1 default program # (Ix91)
X:$0817 (2071) C.S. 1 program execution address pointer
Y:$0817 C.S. 1 program execution status
(Second word returned on **??** command. See X:$0818 for first word.)
(Refer to **??** specification in On-Line Commands for detailed description of bit meanings.)

Bits

0	CIRCLE/SPLINE move mode
1	CCW circular move direction

2	Cutter compensation on
3	Cutter compensation left
4	PVT/SPLINE move mode
5	Segmented move stop request
6	Segmented move acceleration in progress
7	Segmented move in progress
8	Pre Jog move flag
9	Cutter compensation move buffered
10	Cutter compensation move stop request
11	Cutter compensation outside corner
12	Dwell move buffered
13	Synchronous M-variable one-shot
14	End-of-block (/) stop in progress
15	Delayed calculation flag
16	Rotary buffer full
17	In-position true (logical AND of motor bits)
18	Warning following error (logical OR of motor bits)
19	Fatal following error (logical OR of motor bits)
20	Amplifier fault error (logical OR of motor bits)
21	Circle radius error
22	Run time error
23	Program hold (\) in progress

X:$0818 (2072) Coordinate System 1 Status/Control Bits

(First word returned on **??** command. See Y:$0817 for first word.)
(Refer to **??** specification in On-Line Commands for detailed bit meanings.)

0	Program running
1	Single step mode
2	Continuous motion mode
3	Move-specified-by-time mode (not speed)
4	Continuous motion request
5	Radius vector incremental mode
6	A-axis incremental mode
7	A-axis used in feedrate calculations
8	B-axis incremental mode
9	B-axis used in feedrate calculations
10	C-axis incremental mode
11	C-axis used in feedrate calculations
12	U-axis incremental mode
13	U-axis used in feedrate calculations
14	V-axis incremental mode
15	V-axis used in feedrate calculations
16	W-axis incremental mode
17	W-axis used in feedrate calculations
18	X-axis incremental mode
19	X-axis used in feedrate calculations
20	Y-axis incremental mode
21	Y-axis used in feedrate calculations
22	Z-axis incremental mode
23	Z-axis used in feedrate calculations

Y:$0818 (2072) C.S. 1 Motor Definition word

The Motor Definition word is divided into 8 groups of 3 bits. Each bit group reports a motor's assignment as shown below.

Bits	
0-2	Motor 1 assignment
3-5	Motor 2 assignment
6-8	Motor 3 assignment
9-11	Motor 4 assignment
12-14	Motor 5 assignment
15-17	Motor 6 assignment
18-20	Motor 7 assignment
21-23	Motor 8 assignment

Where the motor's assignment is determined by the value of its bit group (3 bits can have a value from 0 to 7).

Value	
0	Not Assigned
1	Assigned to A-axis
2	Assigned to B-axis
3	Assigned to C-axis
4	Assigned to UVW-axes
5	Reserved
6	Reserved
7	Assigned to XYZ-axes

L:$0819 (2073)	C.S. 1 A-Axis desired move position (f.p.)
L:$081A (2074)	C.S. 1 B-Axis desired move position (f.p.)
L:$081B (2075)	C.S. 1 C-Axis desired move position (f.p.)
L:$081C (2076)	C.S. 1 U-Axis desired move position (f.p.)
L:$081D (2077)	C.S. 1 V-Axis desired move position (f.p.)
L:$081E (2078)	C.S. 1 W-Axis desired move position (f.p.)
L:$081F (2079)	C.S. 1 X-Axis desired move position (f.p.)

Mot/CS	1	2	3	4	5	6	7	8
Hex	**[$0820]**	**[$08E0]**	**[$09A0]**	**[$0A60]**	**[$0B20]**	**[$0BE0]**	**[$0CA0]**	**[$0D60]**
Decimal	2080	2272	2462	2656	2848	3040	3232	3424

L:$0820 (2080)	C.S. 1 Y-Axis desired move position (f.p.)
L:$0821 (2081)	C.S. 1 Z-Axis desired move position (f.p.)
L:$0822 (2082)	Motor #1 X/U/A/B/C-Axis coefficient (f.p.)
L:$0823 (2083)	Motor #1 Y/V-Axis coefficient (f.p.)
L:$0824 (2084)	Motor #1 Z/W-Axis coefficient (f.p.)
L:$0825 (2085)	Motor #1 axis offset (f.p.)
Y:$082A (2090)	Motor #1 filtered actual velocity (1/[Ix09*32] cts/servo cycle)
L:$082B (2091)	Motor #1 variable jog position/distance (for J=*, J^*, J:*)
Y:$082E (2094)	Motor #1 integrated current limit (Ix58)
L:$082F (2095)	Motor #1 present integrated current value (X-register in units of Ix58; Y-register fraction)

Mot/CS	1	2	3	4	5	6	7	8
Hex	**[$0830]**	**[$08F0]**	**[$09B0]**	**[$0A70]**	**[$0B30]**	**[$0BF0]**	**[$0CB0]**	**[$0D70]**
Decimal	2096	2288	2480	2672	2864	3056	3248	3440

L:$083E (2110) C.S. 1 cutter compensation radius
L:$083F (2111) C.S. 1 circle radius error limit (Ix96)

Mot/CS	1	2	3	4	5	6	7	8
Hex	[$0840]	[$0900]	[$09C0]	[$0A80]	[$0B40]	[$0C00]	[$0CC0]	[$0D80]
Decimal	2112	2304	2496	2688	2880	3072	3264	3456

D:$0840 (2112) Motor #1 following error (1/[Ix08*32] cts)
L:$0841 (2113) Motor #1 home speed (Ix23) [floating point]
D:$0842 (2114) Motor #1 positive software overtravel limit (Ix13)
D:$0843 (2115) Motor #1 negative software overtravel limit (Ix14)
D:$0844 (2116) Motor #1 position rollover range (Ix27)
X:$0845 (2117) Motor #1 backlash slew rate (Ix85)
Y:$0845 (2117) Motor #1 backlash deadband (from I99)
D:$0846 (2118) Motor #1 saved backlash position
X:$0847 (2119) Motor #1 backlash size (Ix86)
Y:$0847 (2119) Motor #1 present backlash

Mot/CS	1	2	3	4	5	6	7	8
Hex	[$08B0]	[$0970]	[$0A30]	[$0AF0]	[$0BB0]	[$0C70]	[$0D30]	[$0DF0]
Decimal	2224	2416	2608	2800	2992	3184	3376	3568

$08B0 - $08BF C.S. 1 subroutine stack (2224 - 2239)
$08C0 - $097F Motor/Coordinate System 2 Status and Control Registers (equivalent to $0800 - $08BF above) (2240 - 2431)
$0980 - $0A3F Motor/Coordinate System 3 Status and Control Registers (equivalent to $0800 - $08BF above) (2432 - 2623)
$0A40 - $0AFF Motor/Coordinate System 4 Status and Control Registers (equivalent to $0800 - $08BF above) (2624 - 2815)
$0B00 - $0BBF Motor/Coordinate System 5 Status and Control Registers (equivalent to $0800 - $08BF above) (2816 - 3007)
$0BC0 - $0C7F Motor/Coordinate System 6 Status and Control Registers (equivalent to $0800 - $08BF above) (3008 - 3199)
$0C80 - $0D3F Motor/Coordinate System 7 Status and Control Registers (equivalent to $0800 - $08BF above) (3200 - 3391)
$0D40 - $0DFF Motor/Coordinate System 8 Status and Control Registers (equivalent to $0800 - $08BF above) (3392 - 3583)

Buffer Management Registers

X:$0E00 (3584)	1st motion program number (low 16 bits) and entry status (high 8 bits)	
Y:$0E00	1st motion program buffer storage address	
X:$0E01 - X:$0EFF	2nd to 256th program # and entry status	(3585 - 3839)
Y:$0E01 - Y:$0EFF	2nd to 256th program buffer storage address	
X:$0F00 (3840)	PLC 0 execution address	
Y:$0F00	PLC 0 buffer storage address	
X:$0F01 - X:$0F1F	PLC 1 - 31 execution address	
Y:$0F01 - Y:$0F1F	PLC 1 - 31 storage pointer	(3841-3871)
	Bits 0-15 PLC base address 22 PLC disabled	
X:$0F20 (3872)	Data gather buffer start address	
Y:$0F20	Data gather buffer storage address	
(Option GL Lookahead Firmware Only):		
X:$0F20	Lookahead buffer start address	
Y:$0F20	Lookahead buffer storage address	
(The addresses of the following buffer and table addresses are increased by 1.)		
X:$0F21 - X:$0F28	Rotary buffer 1 - 8 start address	(3873-3880)
Y:$0F21 - Y:$0F28	Rotary buffer 1 - 8 storage address	
X:$0F29 (3881)	Transformation matrix buffer start address	
Y:$0F29	Transformation matrix buffer storage address	
X:$0F2A - X:$0F31	Motor 1-8 backlash comp table start address	(3882-3889)
Y:$0F2A - Y:$0F31	Motor 1-8 backlash comp table storage address	
X:$0F32 - X:$0F39	Motor 1-8 torque comp table start address	(3890-3897)
Y:$0F32 - Y:$0F39	Motor 1-8 torque comp table storage address	
X:$0F3A - X:$0F41	Motor 1-8 leadscrew comp table start address	(3898-3905)
Y:$0F3A - Y:$0F41	Motor 1-8 leadscrew comp table storage address	
X:$0F42	Start address of user data buffer (UBUFFER)	
L:$1000 (4096)	Variable P0 (floating point)	
L:$1001 - L:$13FF	Variables P1 - P1023 (floating point)	(4097 - 5119)
L:$1400 (5120)	Variable Q0 (floating point)	
L:$1401 - L:$17FF	Variables Q1 - Q1023 (floating point)	(5121 - 6143)
$1800 (6144)	Start of buffer storage	
$9FFF (40959)	End of buffer storage	
L:$A000 - L:$BBFF	Option 16 Battery-backed parameter memory	(40960 - 48127)

Note:

On boards with flash-backed main memory, registers from X/Y:$BC00 – X/Y:$BFFF are located in the main flash-backed memory if no Option 16 battery-backed memory is present. However, if Option 16 is present, these registers are located instead in the battery-backed memory, and their contents will not be retained through a power-down unless a good battery is present.

X:$BC00 (48128)	User written servo storage	
Y:$BC00 (48128)	Variable M0 definition	
Y:$BC01 - $BFFF	Variable M1 - M1023 definitions	(48129-49151)

PMAC(1) DSPGATE Servo IC Registers

The registers in PMAC(1)'s "DSPGATE" Gate-Array ICs are mapped into the memory space of PMAC's processor. Each DSPGATE contains four consecutively numbered channels; there may be up to 4 DSPGATEs in a PMAC system, for up to 16 channels. Every PMAC contains the first DSPGATE, which has channels 1 through 4. If Option 1 is ordered (not available on PMAC-Lite or Mini-PMAC), the second DSPGATE is provided, which has channels 5 through 8. If an Accessory 24P/V or ACC-51P is attached, the third DSPGATE is provided, which has channels 9 through 12. If Accessory 24P/V Option 1, or a second ACC-51P is attached as well, the fourth DSPGATE is provided, which has channels 13 through 16.

Enc #	1	2	3	4	5	6	7	8
Hex	**[$C000]**	**[$C004]**	**[$C008]**	**[$C00C]**	**[$C010]**	**[$C014]**	**[$C018]**	**[$C01C]**
Decimal	49152	49156	49160	49164	49168	49172	49176	49180
Enc #	9	10	11	12	13	14	15	16
Hex	**[$C020]**	**[$C024]**	**[$C028]**	**[$C02C]**	**[$C030]**	**[$C034]**	**[$C038]**	**[$C03C]**
Decimal	49184	49188	49192	49196	49200	49204	49208	49212

Y:$Cxxx Time between last two encoder counts (SCLK cycles)

X:$Cxxx Encoder Status/Control Bits (Bits 0-15: control; Bits 16-23: status – read-only)

0-3	Decode control *(Encoder I-Variable 0)*
4-7	Position capture control *(Encoder I-Variable 2)*
8-9	Flag select control *(Encoder I-Variable 3)*
10	Count write enable (when = 1, value written to compare register is copied to counter)
11	Compare equal flag latch control
12	Compare-equal output enable
13	EQU output invert enable
14	AENAn output value
15	Digital delay filter disable *(Encoder I-Variable 1)*
16	Compare-equal flag
17	Position-captured flag
18	Count-error flag (latched to 1 on illegal count transition, cleared to 0 on write to counter)
19	Encoder C channel input value
20	HMFLn input value
21	-LIMn input value
22	+LIMn input value
23	FAULTn input value

Enc #	1	2	3	4	5	6	7	8
Hex	**[$C001]**	**[$C005]**	**[$C009]**	**[$C00D]**	**[$C011]**	**[$C015]**	**[$C019]**	**[$C01D]**
Decimal	49153	49157	49161	49165	49169	49173	49177	49181
Enc #	9	10	11	12	13	14	15	16
Hex	**[$C021]**	**[$C025]**	**[$C029]**	**[$C02D]**	**[$C031]**	**[$C035]**	**[$C039]**	**[$C03D]**
Decimal	49185	49189	49193	49197	49201	49205	49209	49213

Y:$Cxxx Time since last encoder count (SCLK cycles)

X:$Cxxx Encoder phase position (counts)

DAC #	2	4	6	8	10	12	14	16
Hex	**[$C002]**	**[$C00A]**	**[$C012]**	**[$C01A]**	**[$C022]**	**[$C02A]**	**[$C032]**	**[$C03A]**
Decimal	49154	49162	49170	49178	49186	49194	49202	49210
Y:$Cxxx	DAC output value (high 16 bits)							

ADC #	1	3	5	7	9	11	13	15
Hex	**[$C006]**	**[$C00E]**	**[$C016]**	**[$C01E]**	**[$C026]**	**[$C02E]**	**[$C036]**	**[$C03E]**
Decimal	49158	49166	49174	49182	49190	49198	49206	49214
Y:$Cxxx	ADC input value (high 16 bits)							

Enc #	1	2	3	4	5	6	7	8
Hex	**[$C002]**	**[$C006]**	**[$C00A]**	**[$C00E]**	**[$C012]**	**[$C016]**	**[$C01A]**	**[$C01E]**
Decimal	49154	49158	49162	49166	49170	49174	49178	49182
Enc #	9	10	11	12	13	14	15	16
Hex	**[$C022]**	**[$C026]**	**[$C02A]**	**[$C02E]**	**[$C032]**	**[$C036]**	**[$C03A]**	**[$C03E]**
Decimal	49186	49190	49194	49198	49202	49206	49210	49214
X:$Cxxx	Encoder servo position (2*counts; LSB is direction)							

DAC #	1	3	5	7	9	11	13	15
Hex	**[$C003]**	**[$C00B]**	**[$C013]**	**[$C01B]**	**[$C023]**	**[$C02B]**	**[$C033]**	**[$C03B]**
Decimal	49155	49163	49171	49179	49187	49195	49203	49211
Y:$Cxxx	DAC output value (high 16 bits)							

ADC #	2	4	6	8	10	12	14	16
Hex	**[$C007]**	**[$C00F]**	**[$C017]**	**[$C01F]**	**[$C027]**	**[$C02F]**	**[$C037]**	**[$C03F]**
Decimal	49159	49167	49175	49183	49191	49199	49207	49215
Y:$Cxxx	ADC input value (high 16 bits)							

Enc #	1	2	3	4	5	6	7	8
Hex	**[$C003]**	**[$C007]**	**[$C00B]**	**[$C00F]**	**[$C013]**	**[$C017]**	**[$C01B]**	**[$C01F]**
Decimal	49155	49159	49163	49167	49171	49175	49179	49183
Enc #	9	10	11	12	13	14	15	16
Hex	**[$C023]**	**[$C027]**	**[$C02B]**	**[$C02F]**	**[$C033]**	**[$C037]**	**[$C03B]**	**[$C03F]**
Decimal	49187	49191	49195	49199	49203	49207	49211	49215
X:$Cxxx	Encoder Capture/Compare position (Capture register is read-only; compare register is write-only.) (When channel's "count-write enable" control bit is set to 1, values written to compare register are copied to active counter.)							

PMAC2 DSPGATE1 Servo IC Registers

Chan #	1	2	3	4	5	6	7	8
Hex	[$C000]	[$C008]	[$C010]	[$C018]	[$C020]	[$C028]	[$C030]	[$C038]
Decimal	49152	49160	49168	49176	49184	49192	49200	49208
Chan #	9	10	11	12	13	14	15	16
Hex	[$C040]	[$C048]	[$C050]	[$C058]	[$C060]	[$C068]	[$C070]	[$C078]
Decimal	49216	49224	49232	49240	49248	49256	49264	49272

Note:

- Channels 1-4, residing in the first DSPGATE1 IC, are present on all PMAC2 boards.
- Channels 5-8, residing in the second DSPGATE1 IC, are present on PMAC2 boards with Option 1.
- Channels 9-12, residing in the third DSPGATE1 IC, are present on PMAC2 systems with a 4-channel PMAC2 ACC-24x2 daughter board.
- Channels 13-16, residing in the fourth DSPGATE1 IC, are present on PMAC2 systems with an 8-channel ACC-24x2 daughter board.

Y:$Cxxx Channel n Time between last two encoder counts (SCLK cycles)

Bits	
0-22	Timer (units of SCLK cycles)
23	Change-of-direction flag

Note: Alternate use if Channel Control Word bit 18 is set to 1 (Rev "D" or newer IC only): Channel n Timer-Based Fractional Count Data (unsigned)

0-11	Compare "B" fractional count (bit 11 = ½-count; bit 10 = ¼-count; etc.)
12-23	Flag-captured fractional count (bit 23 = ½-count; bit 22 = ¼-count; etc.)

X:$Cxxx Channel n Status Word (all bits read-only except bits 7 & 8)

0-2	Capture Hall Effect Device State
3	Invalid demultiplex of C, U, V, & W
4-6	Reserved for future use (reports as 0)
7	Encoder Loss Error (latched at 1 if Tn=Un or Vn=Wn; cleared to 0 by writing 0 to this bit)
8	Encoder Count Error (latched at 1 on illegal encoder transition; cleared to 0 by writing to this bit or resetting counter)
9	Position Compare (EQUn) output value
10	Position-Captured-On-Gated-Index Flag (=0 on read of captured position register, =1 on trigger capture)
11	Position-Captured Flag (on any trigger) (=0 on read of captured position register, =1 on trigger capture)
12	Encoder Channel A (CHAn) Input Value
13	Encoder Channel B (CHBn) Input Value
14	Encoder Channel C (Index, CHCn) Input Value (ungated)
15	Amplifier Fault (FAULTn) Input Value
16	Home Flag (HMFLn) Input Value
17	Positive End Limit (PLIMn) Input Value

18	Negative End Limit (MLIMn) Input Value
19	User Flag (USERn) Input Value
20	FlagWn Input Value
21	FlagVn Input Value
22	FlagUn Input Value
23	FlagTn Input Value

Chan #	1	2	3	4	5	6	7	8
Hex	**[$C001]**	**[$C009]**	**[$C011]**	**[$C019]**	**[$C021]**	**[$C029]**	**[$C031]**	**[$C039]**
Decimal	49153	49161	49169	49177	49185	49193	49201	49209
Chan #	9	10	11	12	13	14	15	16
Hex	**[$C041]**	**[$C049]**	**[$C051]**	**[$C059]**	**[$C061]**	**[$C069]**	**[$C071]**	**[$C079]**
Decimal	49217	49225	49233	49241	49249	49257	49265	49273

Y:$Cxxx Channel n Time since last encoder count (SCLK cycles)
Alternate use if Channel Control Word bit 18 set to 1 (Rev "D" or newer IC only):
Channel n Timer-Based Fractional Count Data (unsigned)

Bits

0-11	Compare "A" fractional count (bit 11 = ½-count; bit 10 = ¼-count; etc.)
12-23	Servo-captured fractional count (bit 23 = ½-count; bit 22 = ¼-count; etc.)

X:$Cxxx Channel n Encoder phase position (counts)

Chan #	1	2	3	4	5	6	7	8
Hex	**[$C002]**	**[$C00A]**	**[$C012]**	**[$C01A]**	**[$C022]**	**[$C02A]**	**[$C032]**	**[$C03A]**
Decimal	49154	49162	49170	49178	49186	49194	49202	49210
Chan #	9	10	11	12	13	14	15	16
Hex	**[$C042]**	**[$C04A]**	**[$C052]**	**[$C05A]**	**[$C062]**	**[$C06A]**	**[$C072]**	**[$C07A]**
Decimal	49218	49226	49234	49242	49250	49258	49266	49274

Y:$Cxxx Channel n Output A Command Value

Bits

8-23	PWM Command Value
6-23	Serial DAC Command Value
0-5	Not Used

X:$Cxxx Channel n Encoder Servo Position Capture Register

0	Direction of last count (0=up, 1=down)
1-23	Position counter (units of counts)

Chan #	1	2	3	4	5	6	7	8
Hex	**[$C003]**	**[$C00B]**	**[$C013]**	**[$C01B]**	**[$C023]**	**[$C02B]**	**[$C033]**	**[$C03B]**
Decimal	49155	49163	49171	49179	49187	49195	49203	49211
Chan #	9	10	11	12	13	14	15	16
Hex	**[$C043]**	**[$C04B]**	**[$C053]**	**[$C05B]**	**[$C063]**	**[$C06B]**	**[$C073]**	**[$C07B]**
Decimal	49219	49227	49235	49243	49251	49259	49267	49275

Y:$Cxxx Channel n Output B Command Value

Bits

8-23 PWM Command Value

6-23 Serial DAC Command Value

0-5 Not used

X:$Cxxx Channel n Flag Position Capture Value; 24 bits, units of counts

Chan #	1	2	3	4	5	6	7	8
Hex	[$C004]	[$C00C]	[$C014]	[$C01C]	[$C024]	[$C02C]	[$C034]	[$C03C]
Decimal	49156	49164	49172	49180	49188	49196	49204	49212
Chan #	9	10	11	12	13	14	15	16
Hex	[$C044]	[$C04C]	[$C054]	[$C05C]	[$C064]	[$C06C]	[$C074]	[$C07C]
Decimal	49220	49228	49236	49244	49252	49260	49268	49276

Y:$Cxxx Channel n Output C Command Value

Bits

8-23 PWM Command Value

0-23 PFM Command Value

X:$Cxxx IC Global Control Word

Channel 1: X:$C004; Channel 5: X:$C024;
Channel 9: X:$C044; Channel 13: X:$C064:
Clock Control Word
(X:$C004 controls channels 1-4; X:$C024 controls channels 5-8.)
(X:$C044 controls channels 9-12; X:$C064 controls channels 13-16.)

(X:$C004 bits 0-11 is I903; X:$C024 bits 0-11 is I907)

0-2 SCLK Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

3-5 PFM Clock Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

6-8 DAC Clock Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

9-11 ADC Clock Frequency Control n (f=39.3216MHz / 2^n, n=0-7)

12 Phase Clock Direction (0=output, 1=input) (This must be 0 in X:$C004; 1 in X:$C024–if 2nd ASIC is used)

13 Servo Clock Direction (0=output, 1=input) (This must be 0 in X:$C004; 1 in X:$C024–if 2nd ASIC is used)

14-15 Reserved for future use (report as zero) (X:$C004 bits 16-19 is I901)

16-19 Phase Clock Frequency Control n (f=MAXPHASE / [n+1], n=0-15) (value in X:$C024 not used) (X:$C004 bits 20-23 is I902)

20-23 Servo Clock Frequency Control n (f=PHASE / [n+1], n=0–15) (value in X:$C024 not used)

Channel 2: X:$C00C; Channel 6: X:$C02C;
Channel 10: X:$C04C; Channel 14: X:$C06C:
DAC Strobe Word, 24 bits
(X:$C00C controls channels 1-4 [I905]; X:$C02C controls channels 5-8 [I909].)
(X:$C04C controls channels 9-12; X:$C06C controls channels 13-16.)
(Shifted out MSB first one bit per DAC_CLK cycle, starting on rising edge of phase clock.)

Channel 3: X:$C014; Channel 7: X:$C034;
Channel 11: X:$C054; Channel 15: X:$C074:
ADC Strobe Word, 24 bits
(Shifted out MSB first one bit per ADC_CLK cycle, starting on rising edge of phase clock.)
Standard mode, bit 0 must be 0 to clear strobe for next cycle; first data bit clocked in ends up in bit 23.
Alternate mode (Rev "D" and newer only), bit 0 set to 1 (strobe cleared automatically); first four data bits clocked in "rolled over" to bits 3-0.
(X:$C014 controls channels 1-4; X:$C034 controls channels 5-8.)
(X:$C054 controls channels 9-12; X:$C074 controls channels 13-16.)
Channel 4: X:$C01C; Channel 8: X:$C03C;
Channel 12: X:$C01C; Channel 16: X:$C03C:
PWM, PFM, MaxPhase Control Word
(X:$C01C controls channels 1-4; X:$C03C controls channels 5-8.)
(X:$C05C controls channels 9-12; X:$C07C controls channels 13-16.)
(X:$C01C bits 0-7 is I904; X:$C03C bits 0-7 is I908.)

0-7	PWM Dead Time (16*PWM CLK cycles) also PFM pulse width (PFM CLK cycles) (X:$C01C bits 8-23 is I900; X:$C03C bits 8-23 is I906.)
8-23	PWM MaxCount Value PWM Frequency = 117.9648 MHz / [4*MaxCount + 6] "MaxPhase" Frequency = 2*PWM Frequency = 117.9648 MHz / [2*MaxCount + 3]

Chan #	1	2	3	4	5	6	7	8
Hex	**[$C005]**	**[$C00D]**	**[$C015]**	**[$C01D]**	**[$C025]**	**[$C02D]**	**[$C035]**	**[$C03D]**
Decimal	49157	49165	49173	49181	49189	49197	49205	49213
Chan #	9	10	11	12	13	14	15	16
Hex	**[$C045]**	**[$C04D]**	**[$C055]**	**[$C05D]**	**[$C065]**	**[$C06D]**	**[$C075]**	**[$C07D]**
Decimal	49221	49229	49237	49245	49253	49261	49269	49277

Y:$Cxxx Channel n ADC A Input Value

Bits

6-23	Serial ADC Value
0-5	Not used (standard ADC strobe mode) ADC header data (alternate ADC strobe mode, "D" rev and newer only)

X:$Cxxx Channel n Control Word

0-3 Encoder Decode Control *(I9n0)*
0000: Pulse and direction decode CW
0001: x1 quadrature decode CW
0010: x2 quadrature decode CW
0011: x4 quadrature decode CW
0100: Pulse and direction decode CCW
0101: x1 quadrature decode CCW
0110: x2 quadrature decode CCW
0111: x4 quadrature decode CCW
1000: Internal pulse and direction decode
1001: (Reserved for future use)
1010: (Reserved for future use)
1011: x6 "hall sensor" decode CW
1100: MLDT timer mode
1101: (Reserved for future use)
1110: (Reserved for future use)
1111: x6 "hall sensor" decode CCW

(Bits 4-7: I9n2)

4-5 Position Capture Control
00: Immediate capture
01: Use encoder index alone
10: Use capture flag alone
11: Use encoder index and capture flag

6 Index Capture Invert Control (0=no inversion, 1=inversion)

7 Flag Capture Invert Control (0=no inversion, 1=inversion)

8-9 Capture Flag Select Control *(I9n3)*
00: Home Flag (HMFLn)
01: Positive End Limit (PLIMn)
10: Negative End Limit (MLIMn)
11: User Flag (USERn)

10 Encoder Counter Reset Control (1=reset)

11 Position Compare Initial State Write Enable

12 Position Compare Initial State Value

13 Position Compare Channel Select *(I9n1)*
(0= use this channel's encoder; 1=use first encoder on IC)

14 AENAn output value

15 Gated Index Select for Position Capture *(I9n4)*
(0=ungated index, 1=gated index)

16 Invert AB for Gated Index *(I9n5)*
(0: Gated Signal=A&B&C; 1: Gated Signal=A/&B/&C)

17 Yaskawa encoder index demultiplex enable ("B" rev and newer only)

18 Hardware fractional count estimation enable *(I9n9)*
("D" rev and newer only)
(0: Normal encoder timer mode;
1: Hardware fractional count estimation ["D" rev and newer only])

19 Invert PFM Direction Control (0=no inversion, 1=invert) *(I9n8)*

(Bits 20-21: I9n7)

20 Invert A & B Output Control (0=no inversion, 1=invert)

21 Invert C Output Control (0=no inversion, 1=invert)

(Bits 22-23: I9n6)

22 Output A & B Mode Select (0=PWM, 1=DAC)

23 Output C Mode Select (0=PWM, 1=PFM)

Chan #	1	2	3	4	5	6	7	8
Hex	**[$C006]**	**[$C00E]**	**[$C016]**	**[$C01E]**	**[$C026]**	**[$C02E]**	**[$C036]**	**[$C03E]**
Decimal	49158	49166	49174	49182	49190	49198	49206	49214
Chan #	9	10	11	12	13	14	15	16
Hex	**[$C046]**	**[$C04E]**	**[$C056]**	**[$C05E]**	**[$C066]**	**[$C06E]**	**[$C076]**	**[$C07E]**
Decimal	49222	49230	49238	49246	49254	49262	49270	49278

Y:$Cxxx Channel n ADC B Input Value

Bits

6-23 Serial ADC Value

0-5 Not used (standard ADC strobe mode) ADC header data (alternate ADC strobe mode, "D" rev and newer

only)

X:$Cxxx Channel n Encoder Compare Auto-increment value (24 bits, units of counts)

Chan #	1	2	3	4	5	6	7	8
Hex	[$C007]	[$C00F]	[$C017]	[$C01F]	[$C027]	[$C02F]	[$C037]	[$C03F]
Decimal	49159	49167	49175	49183	49191	49199	49207	49215
Chan #	9	10	11	12	13	14	15	16
Hex	[$C047]	[$C04F]	[$C057]	[$C05F]	[$C067]	[$C06F]	[$C077]	[$C07F]
Decimal	49223	49231	49239	49247	49255	49263	49271	49279

Y:$Cxxx Channel n Encoder Compare A Value (24 bits, units of counts)

X:$Cxxx Channel n Encoder Compare B Value (24 bits, units of counts)

PMAC2 DSPGATE2 I/O and MACRO Registers

Y:$C080 JI/O Port Data Register (Input or output; when used as general I/O; see Y:$C084)

Bits	
0	I/O00 Data Value
23	I/O23 Data Value

X:$C080 JI/O Port Data Direction Control Register (when used as general I/O; see Y:$C084)

0	I/O00 Direction Control
23	I /O23 Direction Control (All bits: 0=Input; 1=Output)

Y:$C081 JI/O Port Data Register (Input or output; when used as general I/O; see Y:$C085)

0	I/O24 Data Value
7	I/O31 Data Value
8	I/O24 Latched Data Value
15	I/O31 Latched Data Value
16-23	Not used

X:$C081 JI/O Port Data Direction Control Register (when used as general I/O; see Y:$C085)

0	I/O24 Direction Control
7	I/O31 Direction Control (All bits: 0=Input; 1=Output)
8-23	Not used

Y:$C082 JTHW Port Data Register (Input or output; when used as general I/O; see Y:$C086)

0	DAT0 Data Value
7	DAT7 Data Value
8	SEL0 Data Value
15	SEL7 Data Value
16-23	Not used

X:$C082 JTHW Port Data Direction Control Register (when used as general I/O; see Y:$C086)

0	DAT0 Direction Control (Must be 0 to use standard port accessories)
7	DAT7 Direction Control (Must be 0 to use standard port accessories)
8	SEL0 Direction Control (Must be 1 to use standard port accessories)

	15	SEL7 Direction Control (Must be 1 to use standard port accessories) (All bits: 0=Input; 1=Output)
	16-23	Not used
Y:$C083		JDISP, J??? Port Data Register
	0	DISP0 Data Value
	7	DISP7 Data Value
	8	CTRL0 Data Value
	11	CTRL3 Data Value
	12-23	Not used
X:$C083		JDISP, J??? Port Data Direction Control Register
	0	DISP0 Direction Control (Must be 1 for display to function)
	7	DISP7 Direction Control (Must be 1 for display to function)
	8	CTRL0 Direction Control (Must be 1 for ??? to function)
	11	CTRL3 Direction Control (Must be 1 for ??? to function) (All bits: 0=Input; 1=Output)
	12-23	Not used
Y:$C084		JI/O Port Data Type Control Register
	0	I/O00 Data Type Control (0=FlagW1*; 1=I/O00)
	1	I/O01 Data Type Control (0=FlagV1*; 1=I/O01)
	2	I/O02 Data Type Control (0=FlagU1*; 1=I/O02)
	3	I/O03 Data Type Control (0=FlagT1*; 1=I/O03)
	4	I/O04 Data Type Control (0=USER1*; 1=I/O04)
	5	I/O05 Data Type Control (0=MLIM1*; 1=I/O05)
	6	I/O06 Data Type Control (0=PLIM1*; 1=I/O06)
	7	I/O07 Data Type Control (0=HMFL1*; 1=I/O07)
	8	I/O08 Data Type Control (0=PWM_B_BOT1*; 1=I/O08)
	9	I/O09 Data Type Control (0=PWM_B_TOP1*; 1=I/O09)
	10	I/O10 Data Type Control (0=PWM_A_BOT1*; 1=I/O10)
	11	I/O11 Data Type Control (0=PWM_A_TOP1*; 1=I/O11)
	12	I/O12 Data Type Control (0=PWM_B_BOT2*; 1=I/O12)
	13	I/O13 Data Type Control (0=PWM_B_TOP2*; 1=I/O13)
	14	I/O14 Data Type Control (0=PWM_A_BOT2*; 1=I/O14)
	15	I/O15 Data Type Control (0=PWM_A_TOP2*; 1=I/O15)
	16	I/O16 Data Type Control (0=HMFL2*; 1=I/O16)
	17	I/O17 Data Type Control (0=PLIM2*; 1=I/O17)
	18	I/O18 Data Type Control (0=MLIM2*; 1=I/O18)
	19	I/O19 Data Type Control (0=USER2*; 1=I/O19)
	20	I/O20 Data Type Control (0=FlagT2*; 1=I/O20)
	21	I/O21 Data Type Control (0=FlagU2*; 1=I/O21)
	22	I/O22 Data Type Control (0=FlagV2*; 1=I/O22)
	23	I/O23 Data Type Control (0=FlagW2*; 1=I/O23) (All bits: 0=dedicated hardware I/O; 1=general I/O) (All bits must be 1 for JI/O Port to function as general I/O)

X:$C084		JI/O Port Data Inversion Control Register (when used as general I/O; see Y:$C084)
	0	I/O00 Inversion Control
	23	I/O23 Inversion Control (All bits: 0=Non-inverting; 1=Inverting)
Y:$C085		JI/O Port Data Type Control Register
	0	I/O24 Data Type Control
	7	I/O31 Data Type Control (These bits are always 1; there is no alternate mode for these lines)
	8-23	Not used
X:$C085		JI/O Port Data Inversion Control
	0	I/O24 Inversion Control
	7	I/O31 Inversion Control (All bits: 0=Non-inverting; 1=Inverting)
	8-23	Not used
Y:$C086		JTHW Port Data Type Control Register
	0	DAT0 Data Type Control (0=HWC1; 1 =DAT0)
	1	DAT1 Data Type Control (0=HWC2; 1 =DAT1)
	2	DAT2 Data Type Control (0=Fault1*; 1 =DAT2)
	3	DAT3 Data Type Control (0=Fault2*; 1 =DAT3)
	4	DAT4 Data Type Control (0=EQU1*; 1 =DAT4)
	5	DAT5 Data Type Control (0=EQU2*; 1 =DAT5)
	6	DAT6 Data Type Control (0=AENA1*; 1 =DAT6)
	7	DAT7 Data Type Control (0=AENA2*; 1 =DAT7)
	8	SEL0 Data Type Control (0=ADC_STROB*; 1=SEL0)
	9	SEL1 Data Type Control (0=ADC_CLK*; 1=SEL1)
	10	SEL2 Data Type Control (0=ADC_A1*; 1=SEL2)
	11	SEL3 Data Type Control (0=ADC_B1*; 1=SEL3)
	12	SEL4 Data Type Control (0=ADC_A2*; 1=SEL4)
	13	SEL5 Data Type Control (0=ADC_B2*; 1=SEL5)
	14	SEL6 Data Type Control (0=SCLK*; 1=SEL6)
	15	SEL7 Data Type Control (0=SCLK_DIR*; 1=SEL7) (All bits: 0=dedicated hardware I/O; 1=general I/O) (All bits must be 1 for JTHW Port to function as general I/O or with multiplexed accessories)
	16-23	Not used
X:$C086		JTHW Port Data Inversion Control Register (when used as general I/O; see Y:$C086)
	0	DAT0 Inversion Control
	7	DAT7 Inversion Control
	8	SEL0 Inversion Control
	15	SEL7 Inversion Control (All bits: 0=Non-inverting; 1=Inverting) (All bits must be 0 to use standard port accessories)
	16-23	Not used
Y:$C087		JDISP, J??? Port Data Type Control Register
	0	DISP0 Data Type Control
	7	DISP7 Data Type Control
	8	CTRL0 Data Type Control
	11	CTRL3 Data Type Control (These bits are always 1; there is no alternate mode for these pins)
	12-23	Not used
X:$C087		JDISP, J??? Port Data Inversion Control Register

	0	DISP0 Inversion Control
	7	DISP7 Inversion Control
	8	CTRL0 Inversion Control
	11	CTRL3 Inversion Control (All bits: 0=Non-inverting; 1=Inverting) (All bits must be 0 to use standard port accessories)
Y:$C088-$C08B		Not used
X:$C088-$C08B		Not used
Y:$C08C		Pure binary conversion from gray code input on I/O00 to I/O23
X:$C08C		Not used
Y:$C08D		Gray-to-binary conversion bit-length control
	0-3	Bit length of less significant word portion (I/O00 - I/Onn)
	4	=1 specifies 16-bit lower / 8-bit upper conversion
	5-23	Not used
X:$C08D		Not used
Y:$C08E		MACRO Node Enable Control (I996)
	0	Node 0 enable control
	15	Node 15 enable control (0=node disable; 1=node enable)
	16-19	Sync packet slave node number control
	20-23	Master number control (Note: Bits 16-23 not present on prototype PMAC2 boards with prototype "DSPGATE2" ICs; production "DSPGATE2A" ICs have full register)
X:$C08E		Not used
Y:$C08F		MACRO Ring Status and Control
	0	Data overrun error (cleared when read)
	1	Byte violation error (cleared when read)
	2	Packet parity error (cleared when read)
	3	Data underrun error (cleared when read)
	4	Master station enable
	5	Synchronizing master station enable
	6	Sync packet received (cleared when read)
	7	Sync packet phase lock enable
	8	Node 8 master address check disable
	9	Node 9 master address check disable
	10	Node 10 master address check disable
	11	Node 11 master address check disable
	12	Node 12 master address check disable
	13	Node 13 master address check disable
	14	Node 14 master address check disable
	15	Node 15 master address check disable

Note:

On prototype PMAC2 boards with prototype DSPGATE2 ICs, only bits 0, 1, 2 are present; equivalent to bits 4, 1, and 2, respectively, of production boards with DSPGATE2A ICs as listed above.

X:$C08F

DSPGATE2 clock control register
(Bits 0-11 comprise I993)

0-2	Handwheel SCLK* Frequency Control n (f=39.3216MHz / 2^n, n=0-7)
3-5	JHW/PD PFM Clock Frequency Control n (f=39.3216MHz / 2^n, n=0-7)
6-8	Not used
9-11	ADC Clock* Frequency Control n (f=39.3216MHz / 2^n, n=0-7)
12	Phase Clock Direction (0=output, 1=input. This must be 1)
13	Servo Clock Direction (0=output, 1=input. This must be 1)
14-15	Not used (report as zero)
16-19	(I997 – normally used on PMAC2 UltraLite only) Phase Clock* Frequency Control n (f=MAXPHASE* / [n+1], n=0-15)
20-23	(I998 – normally used on PMAC2 UltraLite only) Servo Clock* Frequency Control n (f=PHASE* / [n+1], n=0-15)

Chan #	1*	2*
Hex	**[$C090]**	**[$C098]**
Decimal	49296	49304

Y:$C09x

Handwheel n Time between last two encoder counts (SCLK cycles)

Bits

X:$C09x

Supplementary Channel n* (Handwheel n) Status Word

0-2	Captured Hall Effect Device (UVW) State
3-7	Not used (reports as zero)
8	Encoder Count Error (0 on counter reset, 1 on illegal transition)
9	Position Compare (EQUn*) output value
10	Position-Captured-On-Gated-Index Flag (=0 on read of captured position register, =1 on trigger capture)
11	Position-Captured Flag (on any trigger) (=0 on read of captured position register, =1 on trigger capture)
12	Handwheel 1 Channel A (HWAn) Input Value
13	Handwheel 1 Channel B (HWBn) Input Value
14	Handwheel 1 Channel C (Index, HWCn) Input Value (ungated)
15	Amplifier Fault (FAULTn*) Input Value
16	Home Flag (HMFLn*) Input Value
17	Positive End Limit (PLIMn*) Input Value
18	Negative End Limit (MLIMn*) Input Value
19	User Flag (USERn*) Input Value
20	FlagWn* Input Value
21	FlagVn* Input Value
22	FlagUn* Input Value
23	FlagTn* Input Value

Chan #	1*	2*
Hex	[$C091]	[$C099]
Decimal	49297	49305

Y:$C09x Handwheel n Time Since Last Encoder Count (SCLK cycles)

X:$C09x Handwheel n Phase Position Capture Register (counts)

Chan #	1*	2*
Hex	[$C092]	[$C09A]
Decimal	49298	49306

Y:$C09x Supplementary Channel n* Output A Command Value

Bits

8-23 PWM Command Value (appears on I/O10 & I/O11 if in dedicated mode)

0-5 Not Used

X:$C09x Handwheel n* Servo Position Capture Register

0 Direction of last count (0=up, 1=down)

1-23 Position counter (units of counts)

Chan #	1*	2*
Hex	[$C093]	[$C09B]
Decimal	49299	49307

Y:$C09x Supplementary Channel n* Output B Command Value

Bits

8-23 PWM Command Value (appears on I/O08 & I/O09 if in dedicated mode)

0-5 Not used

X:$C09x Handwheel n Flag Position Capture Value; 24 bits, units of counts

Chan #	1*	2*
Hex	[$C094]	[$C09C]
Decimal	49300	49308

Y:$C09x Supplementary Channel n* Output C Command Value

Bits

8-23 PWM Command Value (appears on PUL1 & DIR1 if in PWM mode)

0-23 PFM Command Value (appears on PUL1 & DIR1 if in PFM mode)

X:$C094 Supplementary ADC Strobe Word, 24 bits (Shifted out MSB first one bit per DAC_CLK cycle, starting on rising edge of phase clock; appears on SEL0 if in dedicated mode.)

X:$C09C Supplementary PWM, PFM, MaxPhase* Control Word

0-7 PWM* Dead Time (16*PWM* CLK cycles) also PFM* pulse width (PFM* CLK cycles)

8-23 PWM* Max Count Value PWM* Frequency = 117.96MHz / [2*(MaxCount+1)] "MaxPhase*" Frequency = 2*PWM* Frequency

Chan #	1*	2*
Hex	[$C095]	[$C09D]
Decimal	49301	49309

Y:$C09x Supplementary Channel n* ADC A Input Value (uses SEL2 in dedicated mode)

Bits

6-23 Serial ADC Value

0-5 Not used

X:$C09x Supplementary Channel n* (Handwheel n) Control Word

0-1 Encoder Decode Control
00: Pulse and direction decode
01: x1 quadrature decode
10: x2 quadrature decode
11: x4 quadrature decode

2-3 Direction & Timer Control
00: Standard timer control, external signal source, no inversion
01: Standard timer control, external signal source, invert direction
10: Standard timer control, internal PFM source, no inversion
11: Alternate timer control, external signal source

4-5 Position Capture Control
00: Software capture (by setting bit 6)
01: Use encoder index alone
10: Use capture flag alone
11: Use encoder index and capture flag

6 Index Capture Invert Control (0=no inversion, 1=inversion)

7 Flag Capture Invert Control (0=no inversion, 1=inversion)

8-9 Capture Flag Select Control
00: Home Flag (HMFLn*)
01: Positive Limit (PLIMn*)
10: Negative Limit (MLIMn*)
11: User Flag (USERn*)

10 Encoder Counter Reset Control (1=reset)

11 Position Compare Initial State Write Enable

12 Position Compare Initial State Value

13 Position Compare Channel Select (0= use this channel's encoder; 1=use first encoder on IC)

14 AENAn* output value

15 Gated Index Select for Position Capture (0=ungated index, 1=gated index)

16 Invert AB for Gated Index (0: Gated Signal=A&B&C; 1: Gated Signal=A/&B/&C)

17-18 Not used (report as 0)

19 Invert PFM Direction Control (0=no inversion, 1=invert)

20 Invert A & B Output Control (0=no inversion, 1=invert)

21 Invert C Output Control (0=no inversion, 1=invert)

22 Not used (reports as 0)
23 Output C Mode Select (0=PWM, 1=PFM)

Chan #	1*	2*
Hex	[$C096]	[$C09E]
Decimal	49302	49310

Y:$C09x Supplementary Channel n* ADC B Input Value (uses SEL3 in dedicated mode)

Bits
6-23 Serial ADC Value
0-5 Not used

X:$C09x Handwheel n Compare Auto-increment value (24 bits, units of counts)

Chan #	1*	2*
Hex	[$C097]	[$C09F]
Decimal	49303	49311

Y:$C09x Handwheel n Compare A Value (24 bits, units of counts)

X:$C09x Handwheel n Compare B Value (24 bits, units of counts)

Y:$C0A0 MACRO Node 0 24-bit command (write) and feedback (read) register

X:$C0A0 MACRO Node 2 24-bit command (write) and feedback (read) register

Y:$C0A1 MACRO Node 0 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$C0A1 MACRO Node 2 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$C0A2 MACRO Node 0 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$C0A2 MACRO Node 2 2nd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$C0A3 MACRO Node 0 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$C0A3 MACRO Node 2 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$C0A4 MACRO Node 1 24-bit command (write) and feedback (read) register

X:$C0A4 MACRO Node 3 24-bit command (write) and feedback (read) register

Y:$C0A5 MACRO Node 1 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$C0A5 MACRO Node 3 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$C0A6 MACRO Node 1 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

X:$C0A6 MACRO Node 3 2nd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Y:$C0A7 MACRO Node 1 3rd 16-bit command (write) and

	feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0A7	MACRO Node 3 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0A8	MACRO Node 4 24-bit command (write) and feedback (read) register
X:$C0A8	MACRO Node 4 24-bit command (write) and feedback (read) register
Y:$C0A9	MACRO Node 6 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0A9	MACRO Node 6 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0AA	MACRO Node 4 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0AA	MACRO Node 6 2nd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0AB	MACRO Node 4 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0AB	MACRO Node 6 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0AC	MACRO Node 5 24-bit command (write) and feedback (read) register
X:$C0AC	MACRO Node 7 24-bit command (write) and feedback (read) register
Y:$C0AD	MACRO Node 5 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0AD	MACRO Node 7 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0AE	MACRO Node 5 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0AE	MACRO Node 7 2nd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0AF	MACRO Node 5 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0AF	MACRO Node 7 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0B0	MACRO Node 8 24-bit command (write) and feedback (read) register
X:$C0B0	MACRO Node 10 24-bit command (write) and feedback (read) register
Y:$C0B1	MACRO Node 8 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0B1	MACRO Node 10 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0B2	MACRO Node 1 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0B2	MACRO Node 10 2nd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0B3	MACRO Node 8 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0B3	MACRO Node 10 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0B4	MACRO Node 9 24-bit command (write) and feedback (read) register
X:$C0B4	MACRO Node 11 24-bit command (write) and feedback (read) register
Y:$C0B5	MACRO Node 9 1st 16-bit command (write) and

	feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0B5	MACRO Node 11 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0B6	MACRO Node 9 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0B6	MACRO Node 11 2nd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0B7	MACRO Node 9 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0B7	MACRO Node 11 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0B8	MACRO Node 12 24-bit command (write) and feedback (read) register
X:$C0B8	MACRO Node 14 24-bit command (write) and feedback (read) register
Y:$C0B9	MACRO Node 12 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0B9	MACRO Node 14 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0BA	MACRO Node 12 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0BA	MACRO Node 14 2nd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0BB	MACRO Node 12 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0BB	MACRO Node 14 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0BC	MACRO Node 13 24-bit command (write) and feedback (read) register
X:$C0BC	MACRO Node 15 24-bit command (write) and feedback (read) register
Y:$C0BD	MACRO Node 13 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0BD	MACRO Node 15 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0BE	MACRO Node 13 1st 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0BE	MACRO Node 15 2nd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
Y:$C0BF	MACRO Node 13 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)
X:$C0BF	MACRO Node 15 3rd 16-bit command (write) and feedback (read) register (bits 8-23; bits 0-7 not used)

Dual-Ported RAM (Option 2 Required)

Note:

Dual-ported RAM addresses are given both as absolute addresses on the PMAC side (with a $ prefix) and as offsets from the base address on the host-computer side (with a "0x" prefix). Detailed information on these functions is given in the manual for the Option 2 DPRAM.

$D000 - $D23F Dedicated DPRAM functions

DPRAM Control Panel Registers

Address	Description
0x0000 (Y:$D000)	Control Panel Motor/Coordinate System Enable Mask (Set bit to enable; PMAC clears on taking action)
	Bit 0: Motor/Coordinate System # 1
	Bit 1: Motor/Coordinate System # 2
	Bit 2: Motor/Coordinate System # 3
	Bit 3: Motor/Coordinate System # 4
	Bit 4: Motor/Coordinate System # 5
	Bit 5: Motor/Coordinate System # 6
	Bit 6: Motor/Coordinate System # 7
	Bit 7: Motor/Coordinate System # 8
	Bits 8-15: Not Used
0x0002 (X:$D000)	Coordinate System Feed Pot Override Enable Mask (Set bit to enable, clear bit to disable)
	Bit 0: Coordinate System # 1
	Bit 1: Coordinate System # 2
	Bit 2: Coordinate System # 3
	Bit 3: Coordinate System # 4
	Bit 4: Coordinate System # 5
	Bit 5: Coordinate System # 6
	Bit 6: Coordinate System # 7
	Bit 7: Coordinate System # 8
	Bits 8-15: Not Used

Control Panel Request Words

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0004	0x0008	0x000C	0x0010	0x0014	0x0018	0x001C	0x0020
PMAC Addr.	Y:$D001	Y:$D002	Y:$D003	Y:$D004	Y:$D005	Y:$D006	Y:$D007	Y:$D008

Bit Format of Request Words

Bit	Request (1 = Action Requested; 0 = No Action Requested)
0-7	(Reserved for Delta Tau Future Use)
8	Jog-Minus (Motor Only) *
9	Jog-Plus (Motor Only) *
10	Pre-Jog (Motor Only)
11	Start (RUN) (Coord. Sys. Only)
12	Step (STEP/QUIT) (Coord. Sys. Only)
13	Stop (ABORT) (Coord. Sys. Only)
14	Home (Motor Only)
15	Feed Hold (HOLD) (Coord. Sys. Only)
* When both Jog-Minus and Jog-Plus are set, motor will stop	

Control Panel Feedrate Override

Coord. Sys. #	1	2	3	4	5	6	7	8
Host Address	0x0006	0x000A	0x000E	0x0012	0x0016	0x001A	0x001E	0x0022
PMAC Addr.	X:$D001	X:$D002	X:$D003	X:$D004	X:$D005	X:$D006	X:$D007	X:$D008

Servo Fixed Data Reporting Buffer

Global Registers for Servo Fixed Data Reporting Buffer

Address	Description
(0x0024) Y:$D009	Host Status to PMAC: Bit 0 = 1 is Host-Busy, reading buffer; = 0 is not busy Bits 1-15 (reserved for future use)
(0x0026) X:$D009	PMAC Status to Host: Bits 0-14 Servo Timer Bit 15 = 1 is PMAC-Busy updating buffer; = 0 is not busy
(0x0028,A) $D00A	Global Status Bits (from Y:$0003) Low 24 Bits (First word returned on ??? command)
(0x002C,E) $D00B	Global Status Bits (from X:$0003) Low 24 bits (Second word returned on ??? command)
(0x0030-46) $D00C-11	Spare Global Var.

Motor-Specific Registers for Servo Fixed Data Reporting Buffer

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0048-0x004E	0x0084-0x008A	0x00C0-0x00C6	0x00FC-0x0102	0x0138-0x013E	0x0174-0x017A	0x01B0-0x01B6	0x01EC-0x01F2
PMAC Addr.	$D012-$D013	$D021-$D022	$D030-$D031	$D03F-$D040	$D04E-$D04F	$D05D-$D05E	$D06C-$D06D	$D07B-$D07C
Source Addr	$0028	$0064	$00A0	$00DC	$0118	$0154	$0190	$01CC
Motor Commanded Position (64 bits; 1/(Ix08*32) counts)								

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0050-0x0056	0x008C-0x0092	0x00C8-0x00CE	0x0104-0x010A	0x0140-0x0146	0x017C-0x0182	0x01B8-0x01BE	0x01F4-0x01FA
PMAC Addr.	$D014-$D015	$D023-$D024	$D032-$D033	$D041-$D042	$D050-$D051	$D05F-$D060	$D06E-$D06F	$D07D-$D07E
Source Addr	$002B	$0067	$00A3	$00DF	$011B	$0157	$0193	$01CF
Motor Actual Position (64 bits; 1/(Ix08*32) counts)								

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0058-0x005E	0x0094-0x009A	0x00D0-0x00D6	0x010C-0x0112	0x0148-0x014E	0x0184-0x018A	0x01C0-0x01C6	0x01FC-0x0202
PMAC Addr.	$D016-$D017	$D025-$D026	$D034-$D035	$D043-$D044	$D052-$D053	$D061-$D062	$D070-$D071	$D07F-$D080
Source Addr.	$002D	$0069	$00A5	$00E1	$011D	$0159	$0195	$01D1
Motor Master Position (64 bits; 1/(Ix07*32) counts of the master; 1/(Ix08*32) motor counts)								

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0060-0x0066	0x009C-0x00A2	0x00D8-0x00DE	0x0114-0x011A	0x0150-0x0156	0x018C-0x0192	0x01C8-0x01CE	0x0204-0x020A
PMAC Addr.	$D018-$D019	$D027-$D028	$D036-$D037	$D045-$D046	$D054-$D055	$D063-$D064	$D072-$D073	$D081-$D082
Source Addr.	$0046	$0082	$00BE	$00FA	$0136	$0172	$01AE	$01EA
Motor Compensation Position (64 bits; 1/(Ix08*32) counts)								

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0068-0x006A	0x00A4-0x00A6	0x00E0-0x00E2	0x011C-0x011E	0x0158-0x015A	0x0194-0x0196	0x01D0-0x01D2	0x020C-0x020E
PMAC Addr.	$D01A	$D029	$D038	$D047	$D056	$D065	$D074	$D083
Source Addr.	X:$003A	X:$0076	X:$00B2	X:$00EE	X:$012A	X:$0166	X:$01A2	X:$01DE
Motor Previous DAC (32 bits; 1/256 DAC bits)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x006C-0x006E	0x00A8-0x00AA	0x00E4-0x00E6	0x0120-0x0122	0x015C-0x015E	0x0198-0x019A	0x01D4-0x01D6	0x0210-0x0212
PMAC Addr.	$D01B	$D02A	$D039	$D048	$D057	$D066	$D075	$D084
Source Addr.	X:$003D	X:$0079	X:$00B5	X:$00F1	X:$012D	X:$0169	X:$01A5	X:$01E1
Motor Servo Status (32 bits; low 24 bits used) 1st word returned on ? command								

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0070 0x0072	0x00AC-0x00AE	0x00E8-0x00EA	0x0124-0x0126	0x0160-0x0162	0x019C-0x019E	0x01D8-0x01DA	0x0214-0x0216
PMAC Addr.	$D01C	$D02B	$D03A	$D049	$D058	$D067	$D076	$D085
Source Addr.	X:$0033	X:$006F	X:$00AB	X:$00E7	X:$0123	X:$015F	X:$019B	X:$01D7
Motor Actual Velocity (1/(Ix09*32) counts per servo cycle)								

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0074-0x0076	0x00B0-0x00B2	0x00EC-0x00EE	0x0128-0x012A	0x0164-0x0166	0x01A0-0x01A2	0x01DC 0x01DE	0x0218-0x021A
PMAC Addr.	$D01D	$D02C	$D03B	$D04A	$D059	$D068	$D077	$D086
Source Addr.	X:$0020	X:$005C	X:$0098	X:$00D4	X:$0110	X:$014C	X:$0188	X:$01C4
Time Left in Move Segment (2*msec)								

Motor #	1	2	3	4	5	6	7	8
Host Address	0x0078-0x007A	0x00B4-0x00B6	0x00F0-0x00F2	0x012C-0x012E	0x0168-0x016A	0x01A4-0x01A6	0x01E0 0x01E2	0x021C-0x021E
PMAC Addr.	$D01E	$D02D	$D03C	$D04B	$D05A	$D069	$D078	$D087
Source Addr.	X:$0029	X:$0065	X:$00A1	X:$00DD	X:$0119	X:$0155	X:$0191	X:$01CD
Handwheel Pointer								

Motor #	1	2	3	4	5	6	7	8
Host Address	0x007C-0x0082	0x00B8-0x00BE	0x00F4-0x00FA	0x0130-0x0136	0x016C-0x0172	0x01A8-0x01AE	0x01E4-0x01EA	0x0220-0x0226
PMAC Addr.	$D01F-$D020	$D02E-$D02F	$D03D-$D03E	$D04C-$D04D	$D05B-$D05C	$D06A-$D06B	$D079-$D07A	$D088-$D089
Spare Registers								

Background Fixed Data Reporting Buffer

Global Registers for Background Fixed Data Buffer:

Address	Description
0x0228 (Y:$D08A)	Buffer Status to Host and PMAC Bit 0: Data-Ready Flag =1 means PMAC done updating buffer =0 means host ready for another update from PMAC Bits 1-15: (Reserved for future use)
0x022A (X:$D08A)	PMAC Servo Timer: Updated at Data Ready Time (from X:$0000)
0x022C,E ($D08B)	Control Panel Hardware Port (from Y:$FFC0)
0x0230,2 ($D08C)	Thumbwheel Hardware Port (from Y:$FFC1)
0x0234,6 ($D08D)	Machine I/O (OPTO) Hardware Port (from Y:$FFC2)
0x0238-4A ($D08E-92)	Spare Global Variable.

Motor/Coordinate System Specific Registers for Background Fixed Data Buffer

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x024C-0x0252	0x02C8-0x02CE	0x0344-0x034A	0x03C0-0x03C6	0x043C-0x0442	0x04B8-0x04BE	0x0534-0x053A	0x05B0-0x05B6
PMAC Addr.	$D093-$D094	$D0B2-$D0B3	$D0D1-$D0D2	$D0F0-$D0F1	$D10F-$D110	$D12E-$D12F	$D14D-$D14E	$D16C-$D16D
Source Addr.	$080B	$08CB	$098B	$0A4B	$0B0B	$0BCB	$0C8B	$0D4B
Motor Target Position (64 bits; 1/(Ix08*32) counts)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0254-0x025A	0x02D0-0x02D6	0x034C-0x0352	0x03C8-0x03CE	0x0444-0x044A	0x04C0-0x04C6	0x053C-0x0542	0x05B8-0x05BE
PMAC Addr.	$D095-$D096	$D0B4-$D0B5	$D0D3-$D0D4	$D0F2-$D0F3	$D111-$D112	$D130-$D131	$D14F-$D150	$D16E-$D16F
Source Addr.	$0813	$08D3	$0993	$0A53	$0B13	$0BD3	$0C93	$0D53
Motor Position Bias (64 bits; 1/(Ix08*32) counts)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x025C-0x025E	0x02D8-0x02DA	0x0354-0x0356	0x03D0-0x03D2	0x044C-0x044E	0x04C8-0x04CA	0x0544-0x0546	0x05C0 0x05C2
PMAC Addr.	$D097	$D0B6	$D0D5	$D0F4	$D113	$D132	$D151	$D170
Source Addr.	Y:$0814	Y:$08D4	Y:$0994	Y:$0A54	Y:$0B14	Y:$0BD4	Y:$0C94	Y:$0D54
Motor Status Word (32 bits; low 24 bits used) 2nd word returned on ? command								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0260-0x0266	0x02DC-0x02E2	0x0358-0x035E	0x03D4-0x03DA	0x0450-0x0456	0x04CC-0x04D2	0x0548-0x054E	0x05C4-0x05CA
PMAC Addr.	$D098-$D099	$D0B7-$D0B8	$D0D6-$D0D7	$D0F5-$D0F6	$D114-$D115	$D133-$D134	$D152-$D153	$D171-$D172
Source Addr	$0818	$08D8	$0998	$0A58	$0B18	$0BD8	$0C98	$0D58
Coordinate System Status/Definition Word (low 32 bits contains Motor Definition Word; high 32 bits contain 1st word returned on ?? command)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0268-0x026E	0x02E4-0x02EA	0x0360-0x0366	0x03DC-0x03E2	0x0458-0x045E	0x04D4-0x04DA	0x0550-0x0556	0x05CC-0x05D2
PMAC Addr.	$D09A-$D09B	$D0B9-$D0BA	$D0D8-$D0D9	$D0F7-$D0F8	$D116-$D117	$D135-$D136	$D154-$D155	$D173-$D174
Source A	$0876	$0936	$09F6	$0AB6	$0B76	$0C36	$0CF6	$0DB6
Source B	$0896	$0956	$0A16	$0AD6	$0B96	$0C56	$0D16	$0DD6
Source C	$0819	$08D9	$0999	$0A59	$0B19	$0BD9	$0C99	$0D59
Coordinate System A-Axis Target Position (User Units) (* Note 1)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0270-0x0276	0x02EC-0x02F2	0x0368-0x036E	0x03E4-0x03EA	0x0460-0x0466	0x04DC-0x04E2	0x0558-0x055E	0x05D4-0x05DA
PMAC Addr.	$D09C-$D09D	$D0BB-$D0BC	$D0DA-$D0DB	$D0F9-$D0FA	$D118-$D119	$D137-$D138	$D156-$D157	$D175-$D176
Source A	$0877	$0937	$09F7	$0AB7	$0B77	$0C37	$0CF7	$0DB7
Source B	$0897	$0957	$0A17	$0AD7	$0B97	$0C57	$0D17	$0DD7
Source C	$081A	$08DA	$099A	$0A5A	$0B1A	$0BDA	$0C9A	$0D5A
Coordinate System B-Axis Target Position (User Units) (* Note 1)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0278-0x027E	0x02F4-0x02FA	0x0370-0x0376	0x03EC-0x03F2	0x0468-0x046E	0x04E4-0x04EA	0x0560-0x0566	0x05DC-0x05E2
PMAC Addr.	$D09E-$D09F	$D0BD-$D0BE	$D0DC-$D0DD	$D0FB-$D0FC	$D11A-$D11B	$D139-$D13A	$D158-$D159	$D177-$D178
Source A	$0878	$0938	$09F8	$0AB8	$0B78	$0C38	$0CF8	$0DB8
Source B	$0898	$0958	$0A18	$0AD8	$0B98	$0C58	$0D18	$0DD8
Source C	$081B	$08DB	$099B	$0A5B	$0B1B	$0BDB	$0C9B	$0D5B
Coordinate System C-Axis Target Position (User Units) (* Note 1)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0280-0x0286	0x02FC-0x0302	0x0378-0x037E	0x03F4-0x03FA	0x0470-0x0476	0x04EC-0x04F2	0x0568-0x056E	0x05E4-0x05EA
PMAC Addr.	$D0A0-$D0A1	$D0BF-$D0C0	$D0DE-$D0DF	$D0FD-$D0FE	$D11C-$D11D	$D13B-$D13C	$D15A-$D15B	$D179-$D17A
Source A	$0879	$0939	$09F9	$0AB9	$0B79	$0C39	$0CF9	$0DB9
Source B	$0899	$0959	$0A19	$0AD9	$0B99	$0C59	$0D19	$0DD9
Source C	$081C	$08DC	$099C	$0A5C	$0B1C	$0BDC	$0C9C	$0D5C
Coordinate System U-Axis Target Position (User Units) (* Note 1)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0288-0x028E	0x0304-0x030A	0x0380-0x0386	0x03FC-0x0402	0x0478-0x047E	0x04F4-0x04FA	0x0570-0x0576	0x05EC-0x05F2
PMAC Addr.	$D0A2-$D0A3	$D0C1-$D0C2	$D0E0-$D0E1	$D0FF-$D100	$D11E-$D11F	$D13D-$D13E	$D15C-$D15D	$D17B-$D17C
Source A	$087A	$093A	$09FA	$0ABA	$0B7A	$0C3A	$0CFA	$0DBA
Source B	$089A	$095A	$0A1A	$0ADA	$0B9A	$0C5A	$0D1A	$0DDA
Source C	$081D	$08DD	$099D	$0A5D	$0B1D	$0BDD	$0C9D	$0D5D
Coordinate System V-Axis Target Position (User Units) (* Note 1)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0290-0x0296	0x030C-0x0312	0x0388-0x038E	0x0404-0x040A	0x0480-0x0486	0x04FC-0x0502	0x0578-0x057E	0x05F4-0x05FA
PMAC Addr.	$D0A4-$D0A5	$D0C3-$D0C4	$D0E2-$D0E3	$D101-$D102	$D120-$D121	$D13F-$D140	$D15E-$D15F	$D17D-$D17E
Source A	$087B	$093B	$09FB	$0ABB	$0B7B	$0C3B	$0CFB	$0DBB
Source B	$089B	$095B	$0A1B	$0ADB	$0B9B	$0C5B	$0D1B	$0DDB
Source C	$081E	$08DE	$099E	$0A5E	$0B1E	$0BDE	$0C9E	$0D5E
Coordinate System W-Axis Target Position (User Units) (* Note 1)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x0298-0x029E	0x0314-0x031A	0x0390-0x0396	0x040C-0x0412	0x0488-0x048E	0x0504-0x050A	0x0580-0x0586	0x05FC-0x0602
PMAC Addr.	$D0A6-$D0A7	$D0C5-$D0C6	$D0E4-$D0E5	$D103-$D104	$D122-$D123	$D141-$D142	$D160-$D161	$D17F-$D180
Source A	$087C	$093C	$09FC	$0ABC	$0B7C	$0C3C	$0CFC	$0DBC
Source B	$089C	$095C	$0A1C	$0ADC	$0B9C	$0C5C	$0D1C	$0DDC
Source C	$081F	$08DF	$099F	$0A5F	$0B1F	$0BDF	$0C9F	$0D5F
Coordinate System X-Axis Target Position (User Units) (* Note 1)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x02A0-0x02A6	0x031C-0x0322	0x0398-0x039E	0x0414-0x041A	0x0490-0x0496	0x050C-0x0512	0x0588-0x058E	0x0604-0x060A
PMAC Addr.	$D0A8-$D0A9	$D0C7-$D0C8	$D0E6-$D0E7	$D105-$D106	$D124-$D125	$D143-$D144	$D162-$D163	$D181-$D182
Source A	$087D	$093D	$09FD	$0ABD	$0B7D	$0C3D	$0CFD	$0DBD
Source B	$089D	$095D	$0A1D	$0ADD	$0B9D	$0C5D	$0D1D	$0DDD
Source C	$0820	$08E0	$09A0	$0A60	$0B20	$0BE0	$0CA0	$0D60
Coordinate System Y-Axis Target Position (User Units) (* Note 1)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x02A8-0x02AE	0x0324-0x032A	0x03A0-0x03A6	0x041C-0x0422	0x0498-0x049E	0x0514-0x051A	0x0590-0x0596	0x060C-0x0612
PMAC Addr.	$D0AA-$D0AB	$D0C9-$D0CA	$D0E8-$D0E9	$D107-$D108	$D126-$D127	$D145-$D146	$D164-$D165	$D183-$D184
Source A	$087E	$093E	$09FE	$0ABE	$0B7E	$0C3E	$0CFE	$0DBE
Source B	$089E	$095E	$0A1E	$0ADE	$0B9E	$0C5E	$0D1E	$0DDE
Source C	$0821	$08E1	$09A1	$0A61	$0B21	$0BE1	$0CA1	$0D61
Coordinate System Z-Axis Target Position (User Units) (* Note 1)								

Note 1:

The following is the logic used in the PMAC to determine which variable will be put in this slot. It is controlled by bits of the coordinate system program execution status word (PSTATUS):

```
If (PSTATUS.7 == 1 && PSTATUS.5 == 0)
    Use Source A
Else
.....If (PSTATUS.9 == 1)
.....          Use Source B
.....Else
.....          Use Source C
.....Endif
Endif
```

PSTATUS.7 is the Segmented move flag (I13 != 0).
PSTATUS.5 is the Segmented move stop flag.
PSTATUS.9 is the Tool Compensation flag.

Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02B0-0x02B2	0x032C-0x032E	0x03A8-0x03AA	0x0424-0x0426	0x04A0-0x04A2	0x051C-0x051E	0x0598-0x059A	0x0614-0x0616
PMAC Addr.	$D0AC	$D0CB	$D0EA	$D109	$D128	$D147	$D166	$D185
Source Addr.	Y:$0817	Y:$08D7	Y:$0997	Y:$0A57	Y:$0B17	Y:$0BD7	Y:$0C97	Y:$0D57
Coordinate System Program Execution Status (32 bits; low 24 bits used) (Second word returned on ?? command)								

Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02B4-0x02B6	0x0330-0x0332	0x03AC-0x03AE	0x0428-0x042A	0x04A4-0x04A6	0x0520-0x0522	0x059C-0x059E	0x0618-0x061A
PMAC Addr.	$D0AD	$D0CC	$D0EB	$D10A	$D129	$D148	$D167	$D186
Source Addr.	Y:$08AE	Y:$096E	Y:$0A2E	Y$0AEE	Y$0BAE	Y:$0C6E	Y:$0D2E	Y$0DEE
Coordinate System Program Lines Remaining (32 bits) (Same value as PR command returns)								

Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02B8-0x02BA	0x0334-0x0336	0x03B0-0x03B2	0x042C-0x042E	0x04A8-0x04AA	0x0524-0x0526	0x05A0-0x05A2	0x061C-0x061E
PMAC Addr.	$D0AE	$D0CD	$D0EC	$D10B	$D12A	$D149	$D168	$D187
Source Addr.	X:$0020	X:$005C	X:$0098	X:$00D4	X:$0110	X:$014C	X:$0188	X:$01C4
Coordinate System Time Remaining in move when I13 > 0 (2*msec)								

Motor/C.S. #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Host Address	0x02BC-0x02BE	0x0338-0x033A	0x03B4-0x03B6	0x0430-0x0432	0x04AC-0x04AE	0x0528-0x052A	0x05A4-0x05A6	0x0620-0x0622
PMAC Addr.	$D0AF	$D0CE	$D0ED	$D10C	$D12B	$D14A	$D169	$D188
Coordinate System Time Remaining in accel/decel when I13 > 0 (2*msec)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x02C0- 0x02C2	0x033C- 0x033E	0x03B8- 0x03BA	0x0434- 0x0436	0x04B0- 0x04B2	0x052C- 0x052E	0x05A8- 0x05AA	0x0624- 0x0626
PMAC Addr.	$D0B0	$D0CF	$D0EE	$D10D	$D12C	$D14B	$D16A	$D189
Coordinate System Program Execution Address Offset (Same value as PE command returns)								

Motor/C.S. #	1	2	3	4	5	6	7	8
Host Address	0x02C4- 0x02C6	0x0340 0x0342	0x03BC- 0x03BE	0x0438- 0x043A	0x04B4- 0x04B6	0x0530- 0x0532	0x05AC- 0x05AE	0x0628- 0x062A
PMAC Addr.	$D0B1	$D0D0	$D0EF	$D10E	$D12D	$D14C	$D16B	$D18A
Source Addr.	Y:$082A	Y:$08EA	Y$09AA	Y$0A6A	Y$0B2A	Y$0BEA	Y$0CAA	Y$0D6
Motor Averaged Actual Velocity (1/[Ix09*32] counts per servo cycle)								

Background Variable Transfer Buffers

PMAC to Host Transfer

Address	Description
0x07E8 **(Y:$D1FA)**	PMAC to HOST (Bit 0 = 1 for single user mode) Data Ready. PMAC done updating buffer - Host must clear for more data.
0x07EA **X:$D1FA**	Servo Timer (Updated at Data Ready Time)
0x07EC **(Y:$D1FB)**	Size of Address Buffer (measured in long integers of 32 bits each)
0x07EE **(X:$D1FB)**	Start of Address Buffer (Ex. $D400; must be $D200 to $DFFD)

Variable Address Buffer Format (2x16-bit words)

X:Mem Bits 15: Data Ready (multi-user mode)	X:Mem Bits 0 - 2: Variable type to read	Y:Mem Variable address	Dual Port Data Length
1 = PMAC data ready 0 = Host request data	0 = PMAC Var. Y:Mem.	PMAC Address of Variable	32 bits
1 = PMAC data ready 0 = Host request data	1 = PMAC Var. Long	PMAC Address of Variable	64 bits
1 = PMAC data ready 0 = Host request data	2 = PMAC Var. X:Mem.	PMAC Address of Variable	32 bits
1 = PMAC data ready 0 = Host request data	4 = Special (Firmware 1.16) PLCC Function Block	PLCC Function Block Number. Y:$9FFF has the base address of the function blocks.	64 bits

Background Variable Data Write Buffer -- Host to PMAC Transfer

Address	Description
0x07E6 **(Y:$D1F5)**	HOST to PMAC Data Transferred. PMAC is updated when cleared. Host must set for another update.
0x07E8 **X:$D1F5**	Starting address of data structure ($D240 - $DFFD).

Variable Address Buffer Format for each Data Structure (6x16-bit)

DPRAM Address	X:Mem	Y:Mem	X:Mem Bit Definitions
$D240	Bits 13 - 15: Special type	PMAC address	0 = PLCC Function Block 1-7 = Reserved for future use
to $DFFD	Bits 8 - 12: Offset	PMAC address	Offset = 0..23. -- This is the starting offset for the read.
	Bits 3 - 7: Width	PMAC address	Width = 0, 1, 4, 8, 12, 16, 20 -- 0 is a 24-bit width.
	Bits 0 - 2: Variable type to write	PLCC Function Block Number	0 = Y register 1 = L register 2 = X register 4 = Special Y register 6 = Special X register
	Upper 16-bits of data 1	Lower 16-bits of data 1	Data to send to PMAC
	Upper 16-bits of data 2	Lower 16-bits of data 2	Data to send to PMAC

Binary Rotary Motion Program Transfer Buffers

Buffer. #	1	2	3	4	5	6	7	8
Host Address	0x07DC	0x07F0	0x07FC	0x0808	0x0814	0x0820	0x082C	0x0838
PMAC Addr.	Y:$D1F7	Y:$D1FC	Y:$D1FF	Y:$D202	Y:$D205	Y:$D208	Y:$D20B	Y:$D20E

PMAC to Host Binary Rotary Buffer Status Word

Bit 15 = 1 represents error; PMAC stops processing commands
Bit 14 = 1 represents PMAC internal rotary buffer full (busy flag)
PMAC Index stops updating.
Bits 7 – 0 represent error code
Code = 1: Internal rotary buffer size = 0 or
DPRAM rotary buffer size =0

These flags are set and reset by the PMAC. The busy flag is set when the PMAC internal rotary buffer is full. This, however, does not necessarily mean that the DPRAM binary rotary buffer is full (see rules). The Busy flag is reset when the PMAC internal rotary buffer is not full or the DPRAM binary rotary buffer is empty.

Buffer. #	1	2	3	4	5	6	7	8
Host Address	0x07DE	0x07F2	0x07FE	0x080A	0x0816	0x0822	0x082EC	0x083A
PMAC Addr.	X:$D1F7	X:$D1FC	X:$D1FF	X:$D202	X:$D205	X:$D208	X:$D20B	X:$D20E

Coordinate System Number and Enable Control

- Bits 0-3 represent Coordinate System #
- Buffer enabled if 0 < (Bits 0-3 value) < 9
- Binary Rotary Buffers; Host to PMAC Transfer

Buffer. #	1	2	3	4	5	6	7	8
Host Address	0x07E0	0x07F4	0x0800	0x080C	0x0818	0x0824	0x0830	0x083C
PMAC Addr.	Y:$D1F7	Y:$D1FD	Y:$D200	Y:$D203	Y:$D206	Y:$D209	Y:$D20C	Y:$D20F

Host Computer Binary Rotary Buffer Index

(In PMAC addresses from start of buffer; each increment is 32-bit word, 4 addresses on host side.)
Host computer updates after loading in program lines

Buffer. #	1	2	3	4	5	6	7	8
Host Address	0x07E2	0x07F6	0x0802	0x080E	0x081A	0x0826	0x0832	0x083E
PMAC Addr.	X:$D1F7	X:$D1FD	X:$D200	X:$D203	X:$D206	X:$D209	X:$D20C	X:$D20F

PMAC Binary Rotary Buffer Index

(In PMAC addresses from start of buffer; each increment is 32-bit word, 4 addresses on host side.)
PMAC updates after reading program lines

Buffer. #	1	2	3	4	5	6	7	8
Host Address	0x07E4	0x07F8	0x0804	0x0810	0x081C	0x0828	0x0834	0x0840
PMAC Addr.	Y:$D1F8	Y:$D1FE	Y:$D201	Y:$D204	Y:$D207	Y:$D20A	Y:$D20D	Y:$D210

Size of Binary Rotary Buffer

(In PMAC addresses; each increment is 32-bit word, 4 addresses on host side.)

Buffer. #	1	2	3	4	5	6	7	8
Host Address	0x07E6	0x07FA	0x0806	0x0812	0x081E	0x082A	0x0836	0x0832
PMAC Addr.	X:$D1F8	X:$D1FE	X:$D201	X:$D204	X:$D207	X:$D20A	X:$D20D	X:$D210

Binary Rotary Buffer PMAC Starting Address

(Address in PMAC's memory map; e.g., $D600, must be >= $D240)

DPRAM Data Gathering Buffer

PMAC to Host Transfer (memory locations set by PMAC)

Address	Description
0x080C (Y:$D23F)	Data Gather Buffer Size.
0x080E (X:$D23F)	PMAC Data Gather Buffer Storage Address. If I45 = 2 and the buffer's end has been reached (this index is greater than or equal to the size), the **DEFINE GATHER** command must be issued again to allow gathering to restart.
0x0810 ($D240)	Start of Data Gather Buffer (not changeable – if other variable-size buffers are used, they must start after end of data gathering buffer.).

Variable-Size Buffers, Open-Use Space

$D240-$DFFF Variable-size buffers, open-use space

VME-Bus Registers (PMAC(1)-VME, PMAC2-VME, PMAC2-VME Ultralite only)

$E000 - $EFFF	Used for VME-bus functions (57344 - 61439)
Y:$E000	VME Mailbox Register 0 (Bits 0-7)
Y:$E001	VME Mailbox Register 1 (Bits 0-7)
Y:$E002	VME Mailbox Register 2 (Bits 0-7)
Y:$E003	VME Mailbox Register 3 (Bits 0-7)
Y:$E004	VME Mailbox Register 4 (Bits 0-7)
Y:$E005	VME Mailbox Register 5 (Bits 0-7)
Y:$E006	VME Mailbox Register 6 (Bits 0-7)
Y:$E007	VME Mailbox Register 7 (Bits 0-7)
Y:$E008	VME Mailbox Register 8 (Bits 0-7)
Y:$E009	VME Mailbox Register 9 (Bits 0-7)
Y:$E00A	VME Mailbox Register A (Bits 0-7)
Y:$E00B	VME Mailbox Register B (Bits 0-7)
Y:$E00C	VME Mailbox Register C (Bits 0-7)
Y:$E00D	VME Mailbox Register D (Bits 0-7)
Y:$E00E	VME Mailbox Register E (Bits 0-7)
Y:$E00F	VME Mailbox Register F (Bits 0-7)

PMAC2 I/O Control Registers

PCI/ISA Bus PMAC2 Versions (PMAC2-PCI, PMAC2-PC, PMAC2-Lite, PMAC2-PC UltraLite):

X/Y:$E800 I/O Buffer IC Direction Control

Bits	
0	OUT0: Buffer direction control for I/O00 to I/O07 on JIO
1	OUT1: Buffer direction control for I/O08 to I/O15 on JIO
2	OUT2: Buffer direction control for I/O16 to I/O23 on JIO
3	OUT3: Buffer direction control for I/O24 to I/O31 on JIO
4	OUT4: Buffer direction control for DAT0 to DAT7 on JTHW
5	OUT5: Buffer direction control for SEL0 to SEL7 on JTHW
6	OUT6: Buffer direction control for DISP0 to DISP7 on JDISP, inverted as R/W- output on JDISP (pin 5)
7	OUT7: Inverted as E output on JDISP

X/Y:$E801 Auxiliary I/O points

0	OUT8: Inverted as RS output on JDISP
1	XIN_1: Jumper E1 input (phase/servo clock direction control)
2	XIN_2: Jumper E2 input (40/60 MHz control)
3	XIN_3: Jumper E3 input (re-initialize on reset)
4	XIN_4: Jumper E4 input
5	XIN_5: Jumper E5 input
6	XIN_6: Jumper E6 input
7	XIN_7: Option 12 ADC convert status

VME Bus PMAC2 Versions (PMAC2-VME, PMAC2-VME UltraLite):

	Bits	
X/Y:$E800		I/O Buffer IC Direction Control
	0	OUT0: Buffer direction control for I/O00 to I/O07 on JIO
	1	OUT1: Buffer direction control for I/O08 to I/O15 on JIO
	2	OUT2: Buffer direction control for I/O16 to I/O23 on JIO
	3	OUT3: Buffer direction control for I/O24 to I/O31 on JIO
X/Y:$E801		I/O Buffer IC Direction Control
	0	OUT8: Inverted as RS output on JDISP
	1	XIN_1: Jumper E1 input; phase/servo clock direction status
	2	XIN_2: Jumper E2 input (40/60 MHz control)
	3	XIN_3: Jumper E3 input (re-initialize on reset)
X/Y:$E802		I/O Buffer IC Direction Control
	0	OUT4: Buffer direction control for DAT0 to DAT7 on JTHW
	1	OUT5: Buffer direction control for SEL0 to SEL7 on JTHW
	2	OUT6: Buffer direction control for DISP0 to DISP7 on JDISP, inverted as R/W- output on JDISP (pin 5)
	3	OUT7: Inverted as E output on JDISP
X/Y:$E803		I/O Buffer IC Direction Control
	0	XIN_4: Jumper E4 input
	1	XIN_5: Jumper E5 input
	2	XIN_6: Jumper E6 input
	3	XIN_7: Option 12 ADC convert status

Inputs and Outputs (PMAC-PC, PMAC-PCI, PMAC-VME, PMAC-Lite, PMAC-PCI Lite only)

	Bits	
Y:$FFC0	0-7	Display & EAROM I/O (dedicated)
(65472)	8	Jog Minus Input (J2-4)
	9	Jog Plus Input (J2-6)
	10	Prejog Input (J2-7)
	11	Start (Run) Input (J2-8)
	12	Step/Quit Input (J2-9)
	13	Stop (Abort) Input (J2-10)
	14	Home Command Input (J2-11)
	15	Feed Hold Input (J2-12)
	16	Motor/CoordSys Select Inp. Bit 0 (J2-3)
	17	Motor/CoordSys Select Inp. Bit 1 (J2-5)
	18	Motor/CoordSys Select Inp. Bit 2 (J2-13)
	19	Motor/CoordSys Select Inp. Bit 3 (J2-14)
	20	E51 Jumper (Reinitialize on power-on)
	21	E50 Jumper: EAROM Write Enable
	22	E49 Jumper: Wait State Control
	23	E48 Jumper: Parity Control
Y:$FFC1	0	Thumbwheel Port Inp. Bit 0 (DAT0:J3-3)
(65473)	1	Thumbwheel Port Inp. Bit 1 (DAT1:J3-5)
	2	Thumbwheel Port Inp. Bit 2 (DAT2:J3-7)
	3	Thumbwheel Port Inp. Bit 3 (DAT3:J3-9)
	4	Thumbwheel Port Inp. Bit 4 (DAT4:J3-11)
	5	Thumbwheel Port Inp. Bit 5 (DAT5:J3-13)
	6	Thumbwheel Port Inp. Bit 6 (DAT6:J3-15)
	7	Thumbwheel Port Inp. Bit 7 (DAT7:J3-17)

	8	Thumbwheel Port Out. Bit 0 (SEL0:J3-4)
	9	Thumbwheel Port Out. Bit 1 (SEL1:J3-6)
	10	Thumbwheel Port Out. Bit 2 (SEL2:J3-8)
	11	Thumbwheel Port Out. Bit 3 (SEL3:J3-10)
	12	Thumbwheel Port Out. Bit 4 (SEL4:J3-12)
	13	Thumbwheel Port Out. Bit 5 (SEL5:J3-14)
	14	Thumbwheel Port Out. Bit 6 (SEL6:J3-16)
	15	Thumbwheel Port Out. Bit 7 (SEL7:J3-18)
	16	Jumper E40: Software Card Address Bit 0
	17	Jumper E41: Software Card Address Bit 1
	18	Jumper E42: Software Card Address Bit 2
	19	Jumper E43: Software Card Address Bit 3
	20	Jumper E44: Baud Rate Select Bit 0
	21	Jumper E45: Baud Rate Select Bit 1
	22	Jumper E46: Baud Rate Select Bit 2
	23	Jumper E47: Baud Rate Select Bit 3
Y:$FFC2	0	Machine Input 1 (MI1) (J5-15)
(65474)	1	Machine Input 2 (MI2) (J5-13)
	2	Machine Input 3 (MI3) (J5-11)
	3	Machine Input 4 (MI4) (J5-9)
	4	Machine Input 5 (MI5) (J5-7)
	5	Machine Input 6 (MI6) (J5-5)
	6	Machine Input 7 (MI7) (J5-3)
	7	Machine Input 8 (MI8) (J5-1)
	8	Machine Output 1 (MO1) (J5-31)
	9	Machine Output 2 (MO2) (J5-29)
	10	Machine Output 3 (MO3) (J5-27)
	11	Machine Output 4 (MO4) (J5-25)
	12	Machine Output 5 (MO5) (J5-23)
	13	Machine Output 6 (MO6) (J5-21)
	14	Machine Output 7 (MO7) (J5-19)
	15	Machine Output 8 (MO8) (J5-17)
	16	D_RS Line (dedicated use)
	17	Read/Write Line (dedicated use)
	18	EACLK (EAROM Clock – dedicated)
	19	ENA422 (Enable RS422 – dedicated)
	20	INPOS (In Position Status Line)
	21	BFUL (Buffer Full Status Line)
	22	EROR (Error Status Line)
	23	F1ER (Following Error Status Line)

Inputs and Outputs (Mini-PMAC, Mini-PMAC-PCI Only)

Y:$FFC0		**J1 (JDISP) Outputs**	
0	DB0	Display Data0	(J1-8)
1	DB1	Display Data1	(J1-7)
2	DB2	Display Data2	(J1-10)
3	DB3	Display Data3	(J1-9)
4	DB4	Display Data 4	(J1-12)
5	DB5	Display Data 5	(J1-11)
6	DB6	Display Data 6	(J1-14)
7	DB7	Display Data 7	(J1-13)
Y:$FFC1		**J3 (JTHW) Inputs**	
0	DAT0	THW Data 0	(J3-3)
1	DAT1	THW Data 1	(J3-5)
2	DAT2	THW Data 2	(J3-7)
3	DAT3	THW Data 3	(J3-9)
4	DAT4	THW Data 4	(J3-11)
5	DAT5	THW Data 5	(J3-13)
6	DAT6	THW Data 6	(J3-15)
7	DAT7	THW Data 7	(J3-17)
Y:$FFC2		**J3 (JTHW) Outputs**	
0	SEL0	THW Select 0	(J3-4)
1	SEL1	THW Select 1	(J3-6)
2	SEL2	THW Select 2	(J3-8)
3	SEL3	THW Select 3	(J3-10)
4	SEL4	THW Select 4	(J3-12)
5	SEL5	THW Select 5	(J3-14)
6	SEL6	THW Select 6	(J3 16)
7	SEL7	THW Select 7	(J3 18)
Y:$FFC3		**J5 (JOPTO) Inputs**	
0	MI1	Machine Input 1	(J5-15)
1	MI2	Machine Input 2	(J5-13)
2	MI3	Machine Input 3	(J5-11)
3	MI4	Machine Input 4	(J5-9)
4	MI5	Machine Input 5	(J5-7)
5	MI6	Machine Input 6	(J5-5)
6	MI7	Machine Input 7	(J5-3)
7	MI8	Machine Input 8	(J5-1)
Y:$FFC4		**J5 (JOPTO) Outputs**	
0	MO1	Machine Output 1	(J5-31)
1	MO2	Machine Output 2	(J5-29)
2	MO3	Machine Output 3	(J5-27)
3	MO4	Machine Output 4	(J5-25)
4	MO5	Machine Output 5	(J5-23)
5	MO6	Machine Output 6	(J5-21)
6	MO7	Machine Output 7	(J5-19)
7	MO8	Machine Output 8	(J5-17)
Y:$FFC5		**Dedicated Use**	
0	ENA422	Serial Enable	
1	RS	Display Control	
2	R/W	Display Control	
3	E	Display Control	
4	E44	Jumper E44	
5	E45	Jumper E45	
6	E46	Jumper E46	

7	E47	Jumper E47
Y:$FFC6		**Dedicated Use**
0	E48	Jumper E48
1	E49	Jumper E49
2	E50	Jumper E50
3	E51	Jumper E51
4	PWR_GUD-	Power Supply Detect
5		(Reserved for future use)
6		(Reserved for future use)
7		(Reserved for future use)

Inputs and Outputs (PMAC1.5-STD Only)

Y:$FFC0		**CPU Board J1 (JDISP) Outputs**	
0	DB0	Display Data 0*	(J1-8)
1	DB1	Display Data 1*	(J1-7)
2	DB2	Display Data 2*	(J1-10)
3	DB3	Display Data 3*	(J1-9)
4	DB4	Display Data 4*	(J1-12)
5	DB5	Display Data 5*	(J1-11)
6	DB6	Display Data 6*	(J1-14)
7	DB7	Display Data 7*	(J1-13)
Y:$FFC1		**CPU Board J3 (JTHW) Outputs**	
0	DAT0	THW Data 0	(J3-3)
1	DAT1	THW Data 1	(J3-5)
2	DAT2	THW Data 2	(J3-7)
3	DAT3	THW Data 3	(J3-9)
4	DAT4	THW Data 4	(J3-11)
5	DAT5	THW Data 5	(J3-13)
6	DAT6	THW Data 6	(J3-15)
7	DAT7	THW Data 7	(J3-17)
Y:$FFC2		**CPU Board J3 (JTHW) Outputs**	
0	SEL0	THW Select 0	(J3-4)
1	SEL1	THW Select 1	(J3-6)
2	SEL2	THW Select 2	(J3-8)
3	SEL3	THW Select 3	(J3-10)
4	SEL4	THW Select 4	(J3-12)
5	SEL5	THW Select 5	(J3-14)
6	SEL6	THW Select 6	(J3 16)
7	SEL7	THW Select 7	(J3 18)
Y:$FFC3		**CPU Board Setup Jumpers**	
0	E40	Jumper E40*	
1	E41	Jumper E41*	
2	E42	Jumper E42*	
3	E43	Jumper E43*	
4	E44	Jumper E44*	
5	E45	Jumper E45*	
6	E46	Jumper E46*	
7	E47	Jumper E47*	
Y:$FFC4		**CPU Board Setup Jumpers and Interrupt Lines**	
0	E48	Jumper E48*	
1	E49	Jumper E49*	
2	E50	Jumper E50*	
3	E51	Jumper E51*	
4	IPOS	In-Position Interrupt	
5	BFUL	Buffer-Full Interrupt	

6	EROR	Fatal F.E. Interrupt	
7	F1ER	Warn. F.E. Interrupt	
Y:$FFC5		**CPU Board Display Control and Panel Indicators**	
0	N.C.	No connect	
1	RS	Read Strobe*	(J1-3)
2	R/W	Read/Write*	(J1-6)
3	E	Enable*	(J1-5)
4	IPLD/	In-Position Indicator	(J3-23)
5	BFLD/	Buffer-Full Indicator	(J3-21)
6	ERLD/	Fatal F.E. Indicator	
7	F1LD/	Warn. F.E. Indicator	
Y:$FFC8		**Axis Board 1 (E93A ON) JOPT port I/O**	
0	MOD00/	Machine I/O 0	(J5-22)
1	MOD01/	Machine I/O 1	(J5-20)
2	MOD02/	Machine I/O 2	(J5-18)
3	MOD03/	Machine I/O 3	(J5-16)
4	MOD04/	Machine I/O 4	(J5-14)
5	MOD05/	Machine I/O 5	(J5-12)
6	MOD06/	Machine I/O 6	(J5-10)
7	MOD07/	Machine I/O 7	(J5-8)
Y:$FFC9		**Axis Board 1 (E93A ON) JOPT port I/O**	
0	MOD08/	Machine I/O 8	(J5-6)
1	MOD09/	Machine I/O 9	(J5-4)
2	MOD10/	Machine I/O 10	(J5 -2)
3	MOD11/	Machine I/O 11	(J5-25)
4	MOD12/	Machine I/O 12	(J5-23)
5	MOD13/	Machine I/O 13	(J5-21)
6	MOD14/	Machine I/O 14	(J5-19)
7	MOD15/	Machine I/O 15	(J5-17)
Y:$FFCA		**Axis Board 1 (E93A ON) JOPT port I/O**	
0	MOD16/	Machine I/O 16	(J5-15)
1	MOD17/	Machine I/O 17	(J5-13)
2	MOD18/	Machine I/O 18	(J5-11)
3	MOD19/	Machine I/O 19	(J5-9)
4	MOD20/	Machine I/O 20	(J5-7)
5	MOD21/	Machine I/O 21	(J5-5)
6	MOD22/	Machine I/O 22	(J5-3)
7	MOD23/	Machine I/O 23	(J5-1)
Y:$FFCB		**Axis Board 1 (E93A ON) JOPT2 port I/O**	
0	MOD24/	Machine I/O 24	(J6-1)
1	MOD25/	Machine I/O 25	(J6-2)
2	MOD26/	Machine I/O 26	(J6-3)
3	MOD27/	Machine I/O 27	(J6-4)
4	MOD28/	Machine I/O 28	(J6-5)
5	MOD29/	Machine I/O 29	(J6-6)
6	MOD30/	Machine I/O 30	(J6-7)
7	MOD31/	Machine I/O 31	(J6-8)
Y:$FFCC		**Axis Board 1 (E93A ON) JPAN port inputs**	
0	JOG-/	JOG IN - DIR.*	(J2-4)
1	JOG+/	JOG IN + DIR.*	(J2-6)
2	PREJ/	RET. TO PREJ*	(J2-7)
3	STRT/	START PROG.*	(J2-8)
4	STEP/	STEP PRGM*	(J2-9)
5	STOP/	STOP PROG.*	(J2-10)

6	HOME/	HOME SEARCH*	(J2-11)
7	HOLD/	HOLD MOTION*	(J2-12)
Y:$FFCD		**Axis Board 1 (E93A ON) JPAN port I/O**	
0	FPD0/	SEL BIT 0*	(J2-3)
1	FPD1/	SEL BIT 1*	(J2-5)
2	FPD2/	SEL BIT 2*	(J2-13)
3	FPD3/	SEL BIT 3*	(J2-14)
4	IPLD/	In Position*	(J2-17)
5	BRLD/	Buffer Request*	(J2-18)
6	ERLD/	Error Indicator*	(J2-19)
7	F1LD/	WarnFollowing Err*	(J2-23)
Y:$FFCF		**Axis Board 1 (E93A ON) Input/Output Control**	
(0=Output OK; 1=Input Only)			
0		MOD00/-MOD07/ Control (Y:$FFC8)	
1		MOD08/-MOD15/ Control (Y:$FFC9)	
2		MOD16/-MOD23/ Control (Y:$FFCA)	
3		MOD24/-MOD31/ Control (Y:$FFCB)	
4		JPAN Switch Control (Y:$FFCC)	
5		JPAN Selector/Indicator Control (Y:$FFCD)	
6		Reserved Control (do not use)	
7		Reserved Control (do not use)	
Y:$FFD0		**Axis Board 2 (E93B ON) JOPT port I/O**	
0	MOD00/	Machine I/O 0	(J5-22)
1	MOD01/	Machine I/O 1	(J5-20)
2	MOD02/	Machine I/O 2	(J5-18)
3	MOD03/	Machine I/O 3	(J5-16)
4	MOD04/	Machine I/O 4	(J5-14)
5	MOD05/	Machine I/O 5	(J5-12)
6	MOD06/	Machine I/O 6	(J5-10)
7	MOD07/	Machine I/O 7	(J5-8)
Y:$FFD1		**Axis Board 2 (E93B ON) JOPT port I/O**	
0	MOD08/	Machine I/O 8	(J5-6)
1	MOD09/	Machine I/O 9	(J5-4)
2	MOD10/	Machine I/O 10	(J5 -2)
3	MOD11/	Machine I/O 11	(J5-25)
4	MOD12/	Machine I/O 12	(J5-23)
5	MOD13/	Machine I/O 13	(J5-21)
6	MOD14/	Machine I/O 14	(J5-19)
7	MOD15/	Machine I/O 15	(J5-17)
Y:$FFD2		**Axis Board 2 (E93B ON) JOPT port I/O**	
0	MOD16/	Machine I/O 16	(J5-15)
1	MOD17/	Machine I/O 17	(J5-13)
2	MOD18/	Machine I/O 18	(J5-11)
3	MOD19/	Machine I/O 19	(J5-9)
4	MOD20/	Machine I/O 20	(J5-7)
5	MOD21/	Machine I/O 21	(J5-5)
6	MOD22/	Machine I/O 22	(J5-3)
7	MOD23/	Machine I/O 23	(J5-1)
Y:$FFD3		**Axis Board 2 (E93B ON) JOPT2 port I/O**	
0	MOD24/	Machine I/O 24	(J6-1)
1	MOD25/	Machine I/O 25	(J6-2)
2	MOD26/	Machine I/O 26	(J6-3)
3	MOD27/	Machine I/O 27	(J6-4)
4	MOD28/	Machine I/O 28	(J6-5)

5	MOD29/	Machine I/O 29	(J6-6)
6	MOD30/	Machine I/O 30	(J6-7)
7	MOD31/	Machine I/O 31	(J6-8)
Y:$FFD4		**Axis Board 2 (E93B ON) JPAN port inputs (general purpose here)**	
0	JOG-/	(General Purpose)	(J2-4)
1	JOG+/	(General Purpose)	(J2-6)
2	PREJ/	(General Purpose)	(J2-7)
3	STRT/	(General Purpose)	(J2-8)
4	STEP/	(General Purpose)	(J2-9)
5	STOP/	(General Purpose)	(J2-10)
6	HOME/	(General Purpose)	(J2-11)
7	HOLD/	(General Purpose)	(J2-12)
Y:$FFD5		**Axis Board 2 (E93B ON) JPAN port I/O (general purpose here)**	
0	FPD0/	(General Purpose)	(J2-3)
1	FPD1/	(General Purpose)	(J2-5)
2	FPD2/	(General Purpose)	(J2-13)
3	FPD3/	(General Purpose)	(J2-14)
4	IPLD/	(General Purpose)	(J2-17)
5	BRLD/	(General Purpose)	(J2-18)
6	ERLD/	(General Purpose)	(J2-19)
7	F1LD/	(General Purpose)	(J2-23)
Y:$FFD7		**Axis Board 2 (E93B ON) Input/Output Control**	
(0=Output OK; 1=Input Only)			
0		MOD00/-MOD07/ Control (Y:$FFD0)	
1		MOD08/-MOD15/ Control (Y:$FFD1)	
2		MOD16/-MOD23/ Control (Y:$FFD2)	
3		MOD24/-MOD31/ Control (Y:$FFD3)	
4		JPAN Switch Control (Y:$FFD4)	
5		JPAN Selector/Indicator Control (Y:$FFD5)	
6		Reserved Control (do not use)	
7		Reserved Control (do not use)	
Y:$FFD8		**Axis Board 3 (E93C ON) JOPT port I/O**	
0	MOD00/	Machine I/O 0	(J5-22)
1	MOD01/	Machine I/O 1	(J5-20)
2	MOD02/	Machine I/O 2	(J5-18)
3	MOD03/	Machine I/O 3	(J5-16)
4	MOD04/	Machine I/O 4	(J5-14)
5	MOD05/	Machine I/O 5	(J5-12)
6	MOD06/	Machine I/O 6	(J5-10)
7	MOD07/	Machine I/O 7	(J5-8)
Y:$FFD9		**Axis Board 3 (E93C ON) JOPT port I/O**	
0	MOD08/	Machine I/O 8	(J5-6)
1	MOD09/	Machine I/O 9	(J5-4)
2	MOD10/	Machine I/O 10	(J5 -2)
3	MOD11/	Machine I/O 11	(J5-25)
4	MOD12/	Machine I/O 12	(J5-23)
5	MOD13/	Machine I/O 13	(J5-21)
6	MOD14/	Machine I/O 14	(J5-19)
7	MOD15/	Machine I/O 15	(J5-17)
Y:$FFDA		**Axis Board 3 (E93C ON) JOPT port I/O**	
0	MOD16/	Machine I/O 16	(J5-15)
1	MOD17/	Machine I/O 17	(J5-13)

2	MOD18/	Machine I/O 18	(J5-11)
3	MOD19/	Machine I/O 19	(J5-9)
4	MOD20/	Machine I/O 20	(J5-7)
5	MOD21/	Machine I/O 21	(J5-5)
6	MOD22/	Machine I/O 22	(J5-3)
7	MOD23/	Machine I/O 23	(J5-1)
Y:$FFDB		**Axis Board 3 (E93C ON) JOPT2 port I/O**	
0	MOD24/	Machine I/O 24	(J6-1)
1	MOD25/	Machine I/O 25	(J6-2)
2	MOD26/	Machine I/O 26	(J6-3)
3	MOD27/	Machine I/O 27	(J6-4)
4	MOD28/	Machine I/O 28	(J6-5)
5	MOD29/	Machine I/O 29	(J6-6)
6	MOD30/	Machine I/O 30	(J6-7)
7	MOD31/	Machine I/O 31	(J6-8)
Y:$FFDC		**Axis Board 3 (E93C ON) JPAN port inputs (general purpose here)**	
0	JOG-/	(General Purpose)	(J2-4)
1	JOG+/	(General Purpose)	(J2-6)
2	PREJ/	(General Purpose)	(J2-7)
3	STRT/	(General Purpose)	(J2-8)
4	STEP/	(General Purpose)	(J2-9)
5	STOP/	(General Purpose)	(J2-10)
6	HOME/	(General Purpose)	(J2-11)
7	HOLD/	(General Purpose)	(J2-12)
Y:$FFDD		**Axis Board 3 (E93C ON) JPAN port I/O (general purpose here)**	
0	FPD0/	(General Purpose)	(J2-3)
1	FPD1/	(General Purpose)	(J2-5)
2	FPD2/	(General Purpose)	(J2-13)
3	FPD3/	(General Purpose)	(J2-14)
4	IPLD/	(General Purpose)	(J2-17)
5	BRLD/	(General Purpose)	(J2-18)
6	ERLD/	(General Purpose)	(J2-19)
7	F1LD/	(General Purpose)	(J2-23)
Y:$FFDF		**Axis Board 3 (E93C ON) Input/Output Control**	
(0=Output OK; 1=Input Only)			
0		MOD00/-MOD07/ Control (Y:$FFD8)	
1		MOD08/-MOD15/ Control (Y:$FFD9)	
2		MOD16/-MOD23/ Control (Y:$FFDA)	
3		MOD24/-MOD31/ Control (Y:$FFDB)	
4		JPAN Switch Control (Y:$FFDC)	
5		JPAN Selector/Indicator Control (Y:$FFDD)	
6		Reserved Control (do not use)	
7		Reserved Control (do not use)	
Y:$FFE0		**Axis Board 4 (E93D ON) JOPT port I/O**	
0	MOD00/	Machine I/O 0	(J5 pin 22)
1	MOD01/	Machine I/O 1	(J5 pin 20)
2	MOD02/	Machine I/O 2	(J5 pin 18)
3	MOD03/	Machine I/O 3	(J5 pin 16)
4	MOD04/	Machine I/O 4	(J5 pin 14)
5	MOD05/	Machine I/O 5	(J5 pin 12)
6	MOD06/	Machine I/O 6	(J5 pin 10)
7	MOD07/	Machine I/O 7	(J5 pin 8)

Y:$FFE1		**Axis Board 4 (E93D ON) JOPT port I/O**	
0	MOD08/	Machine I/O 8	(J5-6)
1	MOD09/	Machine I/O 9	(J5-4)
2	MOD10/	Machine I/O 10	(J5 -2)
3	MOD11/	Machine I/O 11	(J5-25)
4	MOD12/	Machine I/O 12	(J5-23)
5	MOD13/	Machine I/O 13	(J5-21)
6	MOD14/	Machine I/O 14	(J5-19)
7	MOD15/	Machine I/O 15	(J5-17)
Y:$FFE2		**Axis Board 4 (E93D ON) JOPT port I/O**	
0	MOD16/	Machine I/O 16	(J5-15)
1	MOD17/	Machine I/O 17	(J5-13)
2	MOD18/	Machine I/O 18	(J5-11)
3	MOD19/	Machine I/O 19	(J5-9)
4	MOD20/	Machine I/ O 20	(J5-7)
5	MOD21/	Machine I/O 21	(J5-5)
6	MOD22/	Machine I/O 22	(J5-3)
7	MOD23/	Machine I/O 23	(J5-1)
Y:$FFE3		**Axis Board 4 (E93D ON) JOPT2 port I/O**	
0	MOD24/	Machine I/O 24	(J6-1)
1	MOD25/	Machine I/O 25	(J6-2)
2	MOD26/	Machine I/O 26	(J6-3)
3	MOD27/	Machine I/O 27	(J6-4)
4	MOD28/	Machine I/O 28	(J6-5)
5	MOD29/	Machine I/O 29	(J6-6)
6	MOD30/	Machine I/O 30	(J6-7)
7	MOD31/	Machine I/O 31	(J6-8)
Y:$FFE4		**Axis Board 4 (E93D ON) JPAN port inputs (general purpose here)**	
0	JOG-/	(General Purpose)	(J2-4)
1	JOG+/	(General Purpose)	(J2-6)
2	PREJ/	(General Purpose)	(J2-7)
3	STRT/	(General Purpose)	(J2-8)
4	STEP/	(General Purpose)	(J2-9)
5	STOP/	(General Purpose)	(J2-10)
6	HOME/	(General Purpose)	(J2-11)
7	HOLD/	(General Purpose)	(J2-12)
Y:$FFE5		**Axis Board 4 (E93D ON) JPAN port I/O (general purpose here)**	
0	FPD0/	(General Purpose)	(J2-3)
1	FPD1/	(General Purpose)	(J2-5)
2	FPD2/	(General Purpose)	(J2-13)
3	FPD3/	(General Purpose)	(J2-14)
4	IPLD/	(General Purpose)	(J2-17)
5	BREQ/	(General Purpose)	(J2-18)
6	ERLD/	(General Purpose)	(J2-19)
7	F1LD/	(General Purpose)	(J2-23)

Y:$FFE7 **Axis Board 4 (E93D ON) Input/Output Control**

(0=Output OK; 1=Input Only)

0	MOD00/-MOD07/ Control (Y:$FFE0)
1	MOD08/-MOD15/ Control (Y:$FFE1)
2	MOD16/-MOD23/ Control (Y:$FFE2)
3	MOD24/-MOD31/ Control (Y:$FFE3)
4	JPAN Switch Control (Y:$FFE4)
5	JPAN Selector/Indicator Control

	(Y:$FFE5)
6	Reserved Control (do not use)
7	Reserved Control (do not use)

Note:

* All the I/O lines marked with an asterisk in the above table are used by PMAC's firmware for special functions. These lines should not be used for general-purpose I/O.

PMAC2 Option 12/12A Analog-to-Digital Converters

Y:$FFC0 Option 12 Analog-to-Digital Converters (low 12 bits)
Write operation: channel select (Channels 0-7) and mode; Read operation: converted value of selected channel.
Option 12A Analog-to-Digital Converters (high 12 bits)
Write operation: channel select (Channels 8-15) and mode; Read operation: converted value of selected channel.

PMAC(1)-PCI, PMAC(1)-PCI Lite Option 12/12A Analog-to-Digital Converters

Y:$FFC8 Option 12 Analog-to-Digital Converters (low 12 bits)
Write operation: channel select (Channels 0-7) and mode; Read operation: converted value of selected channel.
Option 12A Analog-to-Digital Converters (high 12 bits)
Write operation: channel select (Channels 8-15) and mode; Read operation: converted value of selected channel.

Expansion Port (JEXP) I/O

Y:$FFD0 (65488)	1st Accessory-14 Port A
Y:$FFD1 (65489)	1st Accessory-14 Port B
Y:$FFD2 (65490)	1st Accessory-14 Multimodule
Y:$FFD3 (65491)	1st Accessory-14 Control Word
Y:$FFD8 (65496)	2nd Accessory-14 Port A
Y:$FFD9 (65497)	2nd Accessory-14 Port B
Y:$FFDA (65498)	2nd Accessory-14 Multimodule
Y:$FFDB (65499)	2nd Accessory-14 Control Word
Y:$FFE0 (65504)	3rd Accessory-14 Port A
Y:$FFE1 (65505)	3rd Accessory-14 Port B
Y:$FFE2 (65506)	3rd Accessory-14 Multimodule
Y:$FFE3 (65507)	3rd Accessory-14 Control Word
Y:$FFE8 (65512)	4th Accessory-14 Port A
Y:$FFE9 (65513)	4th Accessory-14 Port B
Y:$FFEA (65514)	4th Accessory-14 Multimodule
Y:$FFEB (65515)	4th Accessory-14 Control Word
Y:$FFF0 (65520)	5th Accessory-14 Port A
Y:$FFF1 (65521)	5th Accessory-14 Port B
Y:$FFF2 (65522)	5th Accessory-14 Multimodule
Y:$FFF3 (65523)	5th Accessory-14 Control Word
Y:$FFF8 (65528)	6th Accessory-14 Port A

Y:$FFF9 (65529)	6th Accessory-14 Port B
Y:$FFFA (65530)	6th Accessory-14 Multimodule
Y:$FFFB (65531)	6th Accessory-14 Control Word

PMAC(1) SUGGESTED M-VARIABLE DEFINITIONS

This file contains a suggested set of M-Variable definitions. These definitions can be very useful for getting access to important I/O and registers. It is not required that you use these particular definitions. Most example programs will use these definitions.

```
CLOSE                          ; To ensure commands are on-line
M0..1023->*                    ; To clear all existing definitions
M0->X:$0,0,24,U                ; Servo cycle counter
; General Purpose inputs and outputs (PMAC-PC,-Lite,-VME)
M1->Y:$FFC2,8,1                ; Machine Output 1
M2->Y:$FFC2,9,1                ; Machine Output 2
M3->Y:$FFC2,10,1               ; Machine Output 3
M4->Y:$FFC2,11,1               ; Machine Output 4
M5->Y:$FFC2,12,1               ; Machine Output 5
M6->Y:$FFC2,13,1               ; Machine Output 6
M7->Y:$FFC2,14,1               ; Machine Output 7
M8->Y:$FFC2,15,1               ; Machine Output 8
M9->Y:$FFC2,8,8,U              ; Machine Outputs 1-8 treated as byte
M11->Y:$FFC2,0,1               ; Machine Input 1
M12->Y:$FFC2,1,1               ; Machine Input 2
M13->Y:$FFC2,2,1               ; Machine Input 3
M14->Y:$FFC2,3,1               ; Machine Input 4
M15->Y:$FFC2,4,1               ; Machine Input 5
M16->Y:$FFC2,5,1               ; Machine Input 6
M17->Y:$FFC2,6,1               ; Machine Input 7
M18->Y:$FFC2,7,1               ; Machine Input 8
M19->Y:$FFC2,0,8,U             ; Machine Inputs 1-8 treated as byte
; Control-Panel Port Input Bits (so can be used as general I/O if I2=1)
; (These definitions valid for PMAC-PC, -Lite, & -VME)
M20->Y:$FFC0,8,1               ; Jog Minus Input
M21->Y:$FFC0,9,1               ; Jog Plus Input
M22->Y:$FFC0,10,1              ; Prejog Input
M23->Y:$FFC0,11,1              ; Start (Run) Input
M24->Y:$FFC0,12,1              ; Step/Quit Input
M25->Y:$FFC0,13,1              ; Stop (Abort) Input
M26->Y:$FFC0,14,1              ; Home Command Input
M27->Y:$FFC0,15,1              ; Feed Hold Input
M28->Y:$FFC0,16,1              ; Motor/C.S. Select Input Bit 0
M29->Y:$FFC0,17,1              ; Motor/C.S. Select Input Bit 1
M30->Y:$FFC0,18,1              ; Motor/C.S. Select Input Bit 2
M31->Y:$FFC0,19,1              ; Motor/C.S. Select Input Bit 3
M32->Y:$FFC0,16,4,C            ; Selected Motor/C.S. Number
; Thumbwheel Port Bits (So they can be used as general-purpose I/O)
; (These definitions valid for PMAC-PC, -Lite, & -VME)
M40->Y:$FFC1,8,1               ; SEL0 Output
M41->Y:$FFC1,9,1               ; SEL1 Output
M42->Y:$FFC1,10,1              ; SEL2 Output
M43->Y:$FFC1,11,1              ; SEL3 Output
```

```
M44->Y:$FFC1,12,1           ; SEL4 Output
M45->Y:$FFC1,13,1           ; SEL5 Output
M46->Y:$FFC1,14,1           ; SEL6 Output
M47->Y:$FFC1,15,1           ; SEL7 Output
M48->Y:$FFC1,8,8,U          ; SEL0-7 Outputs treated as a byte
M50->Y:$FFC1,0,1            ; DAT0 Input
M51->Y:$FFC1,1,1            ; DAT1 Input
M52->Y:$FFC1,2,1            ; DAT2 Input
M53->Y:$FFC1,3,1            ; DAT3 Input
M54->Y:$FFC1,4,1            ; DAT4 Input
M55->Y:$FFC1,5,1            ; DAT5 Input
M56->Y:$FFC1,6,1            ; DAT6 Input
M57->Y:$FFC1,7,1            ; DAT7 Input
M58->Y:$FFC1,0,8,U          ; DAT0-7 Inputs treated as a byte
; Registers associated with Encoder/DAC1 (Usually Motor #1)
M101->X:$C001,0,24,S        ; ENC1 24-bit counter position
M102->Y:$C003,8,16,S        ; DAC1 16-bit analog output
M103->X:$C003,0,24,S        ; ENC1 capture/compare position register
M104->X:$0720,0,24,S        ; ENC1 interpolated position (1/32 ct)
M105->Y:$C006,8,16,S        ; ADC1 16-bit analog input
M106->Y:$C000,0,24,U        ; ENC1 time between counts (SCLK cycles)
M110->X:$C000,10,1          ; ENC1 count-write enable control
M111->X:$C000,11,1          ; EQU1 compare flag latch control
M112->X:$C000,12,1          ; EQU1 compare output enable
M113->X:$C000,13,1          ; EQU1 compare invert enable
M114->X:$C000,14,1          ; AENA1/DIR1 Output
M116->X:$C000,16,1          ; EQU1 compare flag
M117->X:$C000,17,1          ; ENC1 position-captured flag
M118->X:$C000,18,1          ; ENC1 Count-error flag
M119->X:$C000,19,1          ; ENC1 3rd channel input status
M120->X:$C000,20,1          ; HMFL1 input status
M121->X:$C000,21,1          ; -LIM1 input status
M122->X:$C000,22,1          ; +LIM1 input status
M123->X:$C000,23,1          ; FAULT1 input status
; Motor #1 Status Bits
M130->Y:$0814,11,1          ; #1 Stopped-on-position-limit bit
M131->X:$003D,21,1          ; #1 Positive-end-limit-set bit
M132->X:$003D,22,1          ; #1 Negative-end-limit-set bit
M133->X:$003D,13,1          ; #1 Desired-velocity-zero bit
M135->X:$003D,15,1          ; #1 Dwell-in-progress bit
M137->X:$003D,17,1          ; #1 Running-program bit
M138->X:$003D,18,1          ; #1 Open-loop-mode bit
M139->Y:$0814,14,1          ; #1 Amplifier-enabled status bit
M140->Y:$0814,0,1           ; #1 In-position bit
M141->Y:$0814,1,1           ; #1 Warning-following error bit
M142->Y:$0814,2,1           ; #1 Fatal-following-error bit
M143->Y:$0814,3,1           ; #1 Amplifier-fault-error bit
```

```
M145->Y:$0814,10,1              ; #1 Home-complete bit
; Motor #1 Move Registers
M161->D:$0028                   ; #1 Commanded position (1/[Ix08*32] cts)
M162->D:$002B                   ; #1 Actual position (1/[Ix08*32] cts)
M163->D:$080B                   ; #1 Target (end) position (1/[Ix08*32] cts)
M164->D:$0813                   ; #1 Position bias (1/[Ix08*32] cts)
M165->L:$081F                   ; &1 X-axis target position (engineering units)
M166->X:$0033,0,24,S            ; #1 Actual velocity (1/[Ix09*32] cts/cyc)
M167->D:$002D                   ; #1 Present master ((handwheel) pos (1/[Ix07*32] cts
                                ; of master or (1/[Ix08*32] cts of slaved motor)
M168->X:$0045,8,16,S            ; #1 Filter Output (DAC bits)
M169->D:$0046                   ; #1 Compensation correction
M170->D:$0041                   ; #1 Present phase position; includes fraction in Y-register
M171->X:$0041,0,24,S            ; #1 Present phase position (counts*Ix70)
M172->L:$082B                   ; #1 Variable jog position/distance (counts)
M173->Y:$0815,0,24,S            ; #1 Encoder home capture offset (counts)
M174->Y:$082A,24,S              ; #1 Averaged actual velocity (1/[Ix09*32] cts/cyc)
; Coordinate System &1 Status Bits
M180->X:$0818,0,1               ; &1 Program-running bit
M181->Y:$0817,21,1              ; &1 Circle-radius-error bit
M182->Y:$0817,22,1              ; &1 Run-time-error bit
M184->X:$0818,4,1               ; &1 Continuous motion request
M187->Y:$0817,17,1              ; &1 In-position bit (AND of motors)
M188->Y:$0817,18,1              ; &1 Warning-following-error bit (OR)
M189->Y:$0817,19,1              ; &1 Fatal-following-error bit (OR)
M190->Y:$0817,20,1              ; &1 Amp-fault-error bit (OR of motors)
; Motor #1 Axis Definition Registers
M191->L:$0822                   ; #1 X/U/A/B/C-Axis scale factor (cts/unit)
M192->L:$0823                   ; #1 Y/V-Axis scale factor (cts/unit)
M193->L:$0824                   ; #1 Z/W-Axis scale factor (cts/unit)
M194->L:$0825                   ; #1 Axis offset (cts)
; Coordinate System &1 Variables
M197->X:$0806,0,24,S            ; &1 Host commanded time base (I10 units)
M198->X:$0808,0,24,S            ; &1 Present time base (I10 units)
; Registers associated with Encoder/DAC2 (Usually Motor #2)
M201->X:$C005,0,24,S            ; ENC2 24-bit counter position
M202->Y:$C002,8,16,S            ; DAC2 16-bit analog output
M203->X:$C007,0,24,S            ; ENC2 capture/compare position register
M204->X:$0721,0,24,S            ; ENC2 interpolated position (1/32 ct)
M205->Y:$C007,8,16,S            ; ADC2 16-bit analog input
M206->Y:$C004,0,24,U            ; ENC2 time between counts (SCLK cycles)
M210->X:$C004,10,1              ; ENC2 count-write enable control
M211->X:$C004,11,1              ; EQU2 compare flag latch control
M212->X:$C004,12,1              ; EQU2 compare output enable
M213->X:$C004,13,1              ; EQU2 compare invert enable
M214->X:$C004,14,1              ; AENA2/DIR2 Output
M216->X:$C004,16,1              ; EQU2 compare flag
M217->X:$C004,17,1              ; ENC2 position-captured flag
```

```
M218->X:$C004,18,1          ; ENC2 count-error flag
M219->X:$C004,19,1          ; ENC2 3rd channel input status
M220->X:$C004,20,1          ; HMFL2 input status
M221->X:$C004,21,1          ; -LIM2 input status
M222->X:$C004,22,1          ; +LIM2 input status
M223->X:$C004,23,1          ; FAULT2 input status
; Motor #2 Status Bits
M230->Y:$08D4,11,1          ; #2 Stopped-on-position-limit bit
M231->X:$0079,21,1          ; #2 Positive-end-limit-set bit
M232->X:$0079,22,1          ; #2 Negative-end-limit-set bit
M233->X:$0079,13,1          ; #2 Desired-velocity-zero bit
M235->X:$0079,15,1          ; #2 Dwell-in-progress bit
M237->X:$0079,17,1          ; #2 Running-program bit
M238->X:$0079,18,1          ; #2 Open-loop-mode bit
M239->Y:$08D4,14,1          ; #2 Amplifier-enabled status bit
M240->Y:$08D4,0,1           ; #2 In-position bit
M241->Y:$08D4,1,1           ; #2 Warning-following error bit
M242->Y:$08D4,2,1           ; #2 Fatal-following-error bit
M243->Y:$08D4,3,1           ; #2 Amplifier-fault-error bit
M245->Y:$08D4,10,1          ; #2 Home-complete bit
; Motor #2 Move Registers
M261->D:$0064               ; #2 Commanded position (1/[Ix08*32] cts)
M262->D:$0067               ; #2 Actual position (1/[Ix08*32] cts)
M263->D:$08CB               ; #2 Target (end) position (1/[Ix08*32] cts)
M264->D:$08D3               ; #2 Position bias (1/[Ix08*32] cts)
M265->L:$0820               ; &1 Y-axis target position (engineering units)
M266->X:$006F,0,24,S        ; #2 Actual velocity (1/[Ix09*32] cts/cyc)
M267->D:$0069               ; #2 Present master (handwheel) pos (1/[Ix07*32] cts
                            ; of master or (1/[Ix08*32] cts of slaved motor)
M268->X:$0081,8,16,S        ; #2 Filter Output (DAC bits)
M269->D:$0082               ; #2 Compensation correction
M270->D:$007D               ; #2 Present phase position; includes fraction in Y-register
M271->X:$007D,0,24,S        ; #2 Present phase position (counts*Ix70)
M272->L:$08EB               ; #2 Variable jog position/distance (counts)
M273->Y:$08D5,0,24,S        ; #2 Encoder home capture offset (counts)
M274->Y:$08EA,24,S          ; #2 Averaged actual velocity (1/[Ix09*32] cts/cyc)
; Coordinate System &2 Status Bits
M280->X:$08D8,0,1           ; &2 Program-running bit
M281->Y:$08D7,21,1          ; &2 Circle-radius-error bit
M282->Y:$08D7,22,1          ; &2 Run-time-error bit
M284->X:$08D8,4,1           ; &2 Continuous motion request
M287->Y:$08D7,17,1          ; &2 In-position bit (AND of motors)
M288->Y:$08D7,18,1          ; &2 Warning-following-error bit (OR)
M289->Y:$08D7,19,1          ; &2 Fatal-following-error bit (OR)
M290->Y:$08D7,20,1          ; &2 Amp-fault-error bit (OR of motors)
; Motor #2 Axis Definition Registers
M291->L:$08E2               ; #2 X/U/A/B/C-Axis scale factor (cts/unit)
```

```
M292->L:$08E3                ; #2 Y/V-Axis scale factor (cts/unit)
M293->L:$08E4                ; #2 Z/W-Axis scale factor (cts/unit)
M294->L:$08E5                ; #2 Axis offset (cts)
; Coordinate System &2 Variables
M297->X:$08C6,0,24,S         ; &2 Host commanded time base (I10 units)
M298->X:$08C8,0,24,S         ; &2 Present time base (I10 units)
; Registers associated with Encoder/DAC3 (Usually Motor #3)
M301->X:$C009,0,24,S         ; ENC3 24-bit counter position
M302->Y:$C00B,8,16,S         ; DAC3 16-bit analog output
M303->X:$C00B,0,24,S         ; ENC3 capture/compare position register
M304->X:$0722,0,24,S         ; ENC3 interpolated position (1/32 ct)
M305->Y:$C00E,8,16,S         ; ADC3 16-bit analog input
M306->Y:$C008,0,24,U         ; ENC3 time between counts (SCLK cycles)
M310->X:$C008,10,1           ; ENC3 count-write enable control
M311->X:$C008,11,1           ; EQU3 compare flag latch control
M312->X:$C008,12,1           ; EQU3 compare output enable
M313->X:$C008,13,1           ; EQU3 compare invert enable
M314->X:$C008,14,1           ; AENA3/DIR3 Output
M316->X:$C008,16,1           ; EQU3 compare flag
M317->X:$C008,17,1           ; ENC3 position-captured flag
M318->X:$C008,18,1           ; ENC3 count-error flag
M319->X:$C008,19,1           ; ENC3 3rd channel input status
M320->X:$C008,20,1           ; HMFL3 input status
M321->X:$C008,21,1           ; -LIM3 input status
M322->X:$C008,22,1           ; +LIM3 input status
M323->X:$C008,23,1           ; FAULT3 input status
; Motor #3 Status Bits
M330->Y:$0994,11,1           ; #3 Stopped-on-position-limit bit
M331->X:$00B5,21,1           ; #3 Positive-end-limit-set bit
M332->X:$00B5,22,1           ; #3 Negative-end-limit-set bit
M333->X:$00B5,13,1           ; #3 Desired-velocity-zero bit
M335->X:$00B5,15,1           ; #3 Dwell-in-progress bit
M337->X:$00B5,17,1           ; #3 Running-program bit
M338->X:$00B5,18,1           ; #3 Open-loop-mode bit
M339->Y:$0994,14,1           ; #3 Amplifier-enabled status bit
M340->Y:$0994,0,1            ; #3 In-position bit
M341->Y:$0994,1,1            ; #3 Warning-following error bit
M342->Y:$0994,2,1            ; #3 Fatal-following-error bit
M343->Y:$0994,3,1            ; #3 Amplifier-fault-error bit
M345->Y:$0994,10,1           ; #3 Home-complete bit
; Motor #3 Move Registers
M361->D:$00A0                ; #3 Commanded position (1/[Ix08*32] cts)
M362->D:$00A3                ; #3 Actual position (1/[Ix08*32] cts)
M363->D:$098B                ; #3 Target (end) position (1/[Ix08*32] cts)
M364->D:$0993                ; #3 Position bias (1/[Ix08*32] cts)
M365->L:$0821                ; &1 Z-axis target position (engineering units)
M366->X:$00AB,0,24,S         ; #3 Actual velocity (1/[Ix09*32] cts/cyc)
M367->D:$00A5                ; #3 Present master (handwheel) pos (1/[Ix07*32] cts
```

```
                                ; of master or (1/[Ix08*32] cts of slaved motor)
M368->X:$00BD,8,16,S            ; #3 Filter Output (DAC bits)
M369->D:$00BE                   ; #3 Compensation correction
M370->D:$00B9                   ; #3 Present phase position; includes fraction in Y-register
M371->X:$00B9,0,24,S            ; #3 Present phase position (counts*Ix70)
M372->L:$09AB                   ; #3 Variable jog position/distance (counts)
M373->Y:$0995,0,24,S            ; #3 Encoder home capture offset (counts)
M374->Y:$09AA,24,S              ; #3 Averaged actual velocity (1/[Ix09*32] cts/cyc)
; Coordinate System &3 Status Bits
M380->X:$0998,0,1               ; &3 Program-running bit
M381->Y:$0997,21,1              ; &3 Circle-radius-error bit
M382->Y:$0997,22,1              ; &3 Run-time-error bit
M384->X:$0998,4,1               ; &3 Continuous motion request
M387->Y:$0997,17,1              ; &3 In-position bit (AND of motors)
M388->Y:$0997,18,1              ; &3 Warning-following-error bit (OR)
M389->Y:$0997,19,1              ; &3 Fatal-following-error bit (OR)
M390->Y:$0997,20,1              ; &3 Amp-fault-error bit (OR of motors)
; Motor #3 Axis Definition Registers
M391->L:$09A2                   ; #3 X/U/A/B/C-Axis scale factor (cts/unit)
M392->L:$09A3                   ; #3 Y/V-Axis scale factor (cts/unit)
M393->L:$09A4                   ; #3 Z/W-Axis scale factor (cts/unit)
M394->L:$09A5                   ; #3 Axis offset (cts)
; Coordinate System &3 Variables
M397->X:$0986,0,24,S            ; &3 Host commanded time base (I10 units)
M398->X:$0988,0,24,S            ; &3 Present time base (I10 units)
; Registers associated with Encoder/DAC4 (Usually Motor #4)
M401->X:$C00D,0,24,S            ; ENC4 24-bit counter position
M402->Y:$C00A,8,16,S            ; DAC4 16-bit analog output
M403->X:$C00F,0,24,S            ; ENC4 capture/compare position register
M404->X:$0723,0,24,S            ; ENC4 interpolated position (1/32 ct)
M405->Y:$C00F,8,16,S            ; ADC4 16-bit analog input
M406->Y:$C00C,0,24,U            ; ENC4 time between counts (SCLK cycles)
M410->X:$C00C,10,1              ; ENC4 count-write enable control
M411->X:$C00C,11,1              ; EQU4 compare flag latch control
M412->X:$C00C,12,1              ; EQU4 compare output enable
M413->X:$C00C,13,1              ; EQU4 compare invert enable
M414->X:$C00C,14,1              ; AENA4/DIR4 Output
M416->X:$C00C,16,1              ; EQU4 compare flag
M417->X:$C00C,17,1              ; ENC4 position-captured flag
M418->X:$C00C,18,1              ; ENC4 count-error flag
M419->X:$C00C,19,1              ; ENC4 3rd channel input status
M420->X:$C00C,20,1              ; HMFL4 input status
M421->X:$C00C,21,1              ; -LIM4 input status
M422->X:$C00C,22,1              ; +LIM4 input status
M423->X:$C00C,23,1              ; FAULT4 input status
; Motor #4 Status Bits
M430->Y:$0A54,11,1              ; #4 Stopped-on-position-limit bit
M431->X:$00F1,21,1              ; #4 Positive-end-limit-set bit
```

```
M432->X:$00F1,22,1          ; #4 Negative-end-limit-set bit
M433->X:$00F1,13,1          ; #4 Desired-velocity-zero bit
M435->X:$00F1,15,1          ; #4 Dwell-in-progress bit
M437->X:$00F1,17,1          ; #4 Running-program bit
M438->X:$00F1,18,1          ; #4 Open-loop-mode bit
M439->Y:$0A54,14,1          ; #4 Amplifier-enabled status bit
M440->Y:$0A54,0,1           ; #4 In-position bit
M441->Y:$0A54,1,1           ; #4 Warning-following error bit
M442->Y:$0A54,2,1           ; #4 Fatal-following-error bit
M443->Y:$0A54,3,1           ; #4 Amplifier-fault-error bit
M445->Y:$0A54,10,1          ; #4 Home-complete bit
; Motor #4 Move Registers
M461->D:$00DC               ; #4 Commanded position (1/[Ix08*32] cts)
M462->D:$00DF               ; #4 Actual position (1/[Ix08*32] cts)
M463->D:$0A4B               ; #4 Target (end) position (1/[Ix08*32] cts)
M464->D:$0A53               ; #4 Position bias (1/[Ix08*32] cts)
M465->L:$0819               ; &1 A-axis target position (engineering units)
M466->X:$00E7,0,24,S        ; #4 Actual velocity (1/[Ix09*32] cts/cyc)
M467->D:$00E1               ; #4 Present master (handwheel) pos (1/[Ix07*32] cts
                            ; of master or (1/[Ix08*32] cts of slaved motor)
M468->X:$00F9,8,16,S        ; #4 Filter Output (DAC bits)
M469->D:$00FA               ; #4 Compensation correction
M470->D:$00F5               ; #4 Present phase position; includes fraction in Y-register
M471->X:$00F5,0,24,S        ; #4 Present phase position (counts*Ix70)
M472->L:$0A6B               ; #4 Variable jog position/distance (counts)
M473->Y:$0A55,0,24,S        ; #4 Encoder home capture offset (counts)
M474->Y:$0A6A,24,S          ; #4 Averaged actual velocity (1/[Ix09*32] cts/cyc)
; Coordinate System &4 Status Bits
M480->X:$0A58,0,1           ; &4 Program-running bit
M481->Y:$0A57,21,1          ; &4 Circle-radius-error bit
M482->Y:$0A57,22,1          ; &4 Run-time-error bit
M484->X:$0A58,4,1           ; &4 Continuous motion request
M487->Y:$0A57,17,1          ; &4 In-position bit (AND of motors)
M488->Y:$0A57,18,1          ; &4 Warning-following-error bit (OR)
M489->Y:$0A57,19,1          ; &4 Fatal-following-error bit (OR)
M490->Y:$0A57,20,1          ; &4 Amp-fault-error bit (OR of motors)
; Motor #4 Axis Definition Registers
M491->L:$0A62               ; #4 X/U/A/B/C-Axis scale factor (cts/unit)
M492->L:$0A63               ; #4 Y/V-Axis scale factor (cts/unit)
M493->L:$0A64               ; #4 Z/W-Axis scale factor (cts/unit)
M494->L:$0A65               ; #4 Axis offset (cts)
; Coordinate System &4 Variables
M497->X:$0A46,0,24,S        ; &4 Host commanded time base (I10 units)
M498->X:$0A48,0,24,S        ; &4 Present time base (I10 units)
; Registers associated with Encoder/DAC5 (Usually Motor #5)
M501->X:$C011,0,24,S        ; ENC5 24-bit counter position
M502->Y:$C013,8,16,S        ; DAC5 16-bit analog output
M503->X:$C013,0,24,S        ; ENC5 capture/compare position register
```

```
M504->X:$0724,0,24,S          ; ENC5 interpolated position (1/32 ct)
M505->Y:$C016,8,16,S          ; ADC5 16-bit analog input
M506->Y:$C010,0,24,U          ; ENC5 time between counts (SCLK cycles)
M510->X:$C010,10,1            ; ENC5 count-write enable control
M511->X:$C010,11,1            ; EQU5 compare flag latch control
M512->X:$C010,12,1            ; EQU5 compare output enable
M513->X:$C010,13,1            ; EQU5 compare invert enable
M514->X:$C010,14,1            ; AENA5/DIR5 Output
M516->X:$C010,16,1            ; EQU5 compare flag
M517->X:$C010,17,1            ; ENC5 position-captured flag
M518->X:$C010,18,1            ; ENC5 count-error flag
M519->X:$C010,19,1            ; ENC5 3rd channel input status
M520->X:$C010,20,1            ; HMFL5 input status
M521->X:$C010,21,1            ; -LIM5 input status
M522->X:$C010,22,1            ; +LIM5 input status
M523->X:$C010,23,1            ; FAULT5 input status
; Motor #5 Status Bits
M530->Y:$0B14,11,1            ; #5 Stopped-on-position-limit bit
M531->X:$012D,21,1            ; #5 Positive-end-limit-set bit
M532->X:$012D,22,1            ; #5 Negative-end-limit-set bit
M533->X:$012D,13,1            ; #5 Desired-velocity-zero bit
M535->X:$012D,15,1            ; #5 Dwell-in-progress bit
M537->X:$012D,17,1            ; #5 Running-program bit
M538->X:$012D,18,1            ; #5 Open-loop-mode bit
M539->Y:$0B14,14,1            ; #5 Amplifier-enabled status bit
M540->Y:$0B14,0,1             ; #5 In-position bit
M541->Y:$0B14,1,1             ; #5 Warning-following error bit
M542->Y:$0B14,2,1             ; #5 Fatal-following-error bit
M543->Y:$0B14,3,1             ; #5 Amplifier-fault-error bit
M545->Y:$0B14,10,1            ; #5 Home-complete bit
; Motor #5 Move Registers
M561->D:$0118                 ; #5 Commanded position (1/[Ix08*32] cts)
M562->D:$011B                 ; #5 Actual position (1/[Ix08*32] cts)
M563->D:$0B0B                 ; #5 Target (end) position (1/[Ix08*32] cts)
M564->D:$0B13                 ; #5 Position bias (1/[Ix08*32] cts)
M565->L:$081A                 ; &1 B-axis target position (engineering units)
M566->X:$0123,0,24,S          ; #5 Actual velocity (1/[Ix09*32] cts/cyc)
M567->D:$011D                 ; #5 Present master (handwheel) pos (1/[Ix07*32] cts
                              ; of master or (1/[Ix08*32] cts of slaved motor)
M568->X:$0135,8,16,S          ; #5 Filter Output (DAC bits)
M569->D:$0136                 ; #5 Compensation correction
M570->D:$0131                 ; #5 Present phase position; includes fraction in Y-register
M571->X:$0131,0,24,S          ; #5 Present phase position (counts*Ix70)
M572->L:$0B2B                 ; #5 Variable jog position/distance (counts)
M573->Y:$0B15,0,24,S          ; #5 Encoder home capture offset (counts)
M574->Y:$0B2A,24,S            ; #5 Averaged actual velocity (1/[Ix09*32] cts/cyc)
; Coordinate System &5 Status Bits
M580->X:$0B18,0,1             ; &5 Program-running bit
```

```
M581->Y:$0B17,21,1          ; &5 Circle-radius-error bit
M582->Y:$0B17,22,1          ; &5 Run-time-error bit
M584->X:$0B18,4,1           ; &5 Continuous motion request
M587->Y:$0B17,17,1          ; &5 In-position bit (AND of motors)
M588->Y:$0B17,18,1          ; &5 Warning-following-error bit (OR)
M589->Y:$0B17,19,1          ; &5 Fatal-following-error bit (OR)
M590->Y:$0B17,20,1          ; &5 Amp-fault-error bit (OR of motors)
; Motor #5 Axis Definition Registers
M591->L:$0B22               ; #5 X/U/A/B/C-Axis scale factor (cts/unit)
M592->L:$0B23               ; #5 Y/V-Axis scale factor (cts/unit)
M593->L:$0B24               ; #5 Z/W-Axis scale factor (cts/unit)
M594->L:$0B25               ; #5 Axis offset (cts)
; Coordinate System &5 Variables
M597->X:$0B06,0,24,S        ; &5 Host commanded time base (I10 units)
M598->X:$0B08,0,24,S        ; &5 Present time base (I10 units)
; Registers associated with Encoder/DAC6 (Usually Motor #6)
M601->X:$C015,0,24,S        ; ENC6 24-bit counter position
M602->Y:$C012,8,16,S        ; DAC6 16-bit analog output
M603->X:$C017,0,24,S        ; ENC6 capture/compare position register
M604->X:$0725,0,24,S        ; ENC6 interpolated position (1/32 ct)
M605->Y:$C017,8,16,S        ; ADC6 16-bit analog input
M606->Y:$C014,0,24,U        ; ENC6 time between counts (SCLK cycles)
M610->X:$C014,10,1          ; ENC6 count-write enable control
M611->X:$C014,11,1          ; EQU6 compare flag latch control
M612->X:$C014,12,1          ; EQU6 compare output enable
M613->X:$C014,13,1          ; EQU6 compare invert enable
M614->X:$C014,14,1          ; AENA6/DIR6 Output
M616->X:$C014,16,1          ; EQU6 compare flag
M617->X:$C014,17,1          ; ENC6 position-captured flag
M618->X:$C014,18,1          ; ENC6 count-error flag
M619->X:$C014,19,1          ; ENC6 3rd channel input status
M620->X:$C014,20,1          ; HMFL6 input status
M621->X:$C014,21,1          ; -LIM6 input status
M622->X:$C014,22,1          ; +LIM6 input status
M623->X:$C014,23,1          ; FAULT6 input status
; Motor #6 Status Bits
M630->Y:$0BD4,11,1          ; #6 Stopped-on-position-limit bit
M631->X:$0169,21,1          ; #6 Positive-end-limit-set bit
M632->X:$0169,22,1          ; #6 Negative-end-limit-set bit
M633->X:$0169,13,1          ; #6 Desired-velocity-zero bit
M635->X:$0169,15,1          ; #6 Dwell-in-progress bit
M637->X:$0169,17,1          ; #6 Running-program bit
M638->X:$0169,18,1          ; #6 Open-loop-mode bit
M639->Y:$0BD4,14,1          ; #6 Amplifier-enabled status bit
M640->Y:$0BD4,0,1           ; #6 In-position bit
M641->Y:$0BD4,1,1           ; #6 Warning-following error bit
M642->Y:$0BD4,2,1           ; #6 Fatal-following-error bit
M643->Y:$0BD4,3,1           ; #6 Amplifier-fault-error bit
```

```
M645->Y:$0BD4,10,1              ; #6 Home-complete bit
; Motor #6 Move Registers
M661->D:$0154                   ; #6 Commanded position (1/[Ix08*32] cts)
M662->D:$0157                   ; #6 Actual position (1/[Ix08*32] cts)
M663->D:$0BCB                   ; #6 Target (end) position (1/[Ix08*32] cts)
M664->D:$0BD3                   ; #6 Position bias (1/[Ix08*32] cts)
M665->L:$081B                   ; &1 C-axis target position (engineering units)
M666->X:$015F,0,24,S            ; #6 Actual velocity (1/[Ix09*32] cts/cyc)
M667->D:$0159                   ; #6 Present master (handwheel) pos (1/[Ix07*32] cts
                                ; of master or (1/[Ix08*32] cts of slaved motor)
M668->X:$0171,8,16,S            ; #6 Filter Output (DAC bits)
M669->D:$0172                   ; #6 Compensation correction
M670->D:$016D                   ; #6 Present phase position; includes fraction in Y-register
M671->X:$016D,0,24,S            ; #6 Present phase position (counts*Ix70)
M672->L:$0BEB                   ; #6 Variable jog position/distance (counts)
M673->Y:$0BD5,0,24,S            ; #6 Encoder home capture offset (counts)
M674->Y:$0BEA,24,S              ; #6 Averaged actual velocity (1/[Ix09*32] cts/cyc)
; Coordinate System &6 Status Bits
M680->X:$0BD8,0,1               ; &6 Program-running bit
M681->Y:$0BD7,21,1              ; &6 Circle-radius-error bit
M682->Y:$0BD7,22,1              ; &6 Run-time-error bit
M684->X:$0BD8,4,1               ; &6 Continuous motion request
M687->Y:$0BD7,17,1              ; &6 In-position bit (AND of motors)
M688->Y:$0BD7,18,1              ; &6 Warning-following-error bit (OR)
M689->Y:$0BD7,19,1              ; &6 Fatal-following-error bit (OR)
M690->Y:$0BD7,20,1              ; &6 Amp-fault-error bit (OR of motors)
; Motor #6 Axis Definition Registers
M691->L:$0BE2                   ; #6 X/U/A/B/C-Axis scale factor (cts/unit)
M692->L:$0BE3                   ; #6 Y/V-Axis scale factor (cts/unit)
M693->L:$0BE4                   ; #6 Z/W-Axis scale factor (cts/unit)
M694->L:$0BE5                   ; #6 Axis offset (cts)
; Coordinate System &6 Variables
M697->X:$0BC6,0,24,S            ; &6 Host commanded time base (I10 units)
M698->X:$0BC8,0,24,S            ; &6 Present time base (I10 units)
; Registers associated with Encoder/DAC7 (Usually Motor #7)
M701->X:$C019,0,24,S            ; ENC7 24-bit counter position
M702->Y:$C01B,8,16,S            ; DAC7 16-bit analog output
M703->X:$C01B,0,24,S            ; ENC7 capture/compare position register
M704->X:$0726,0,24,S            ; ENC7 interpolated position (1/32 ct)
M705->Y:$C01E,8,16,S            ; ADC7 16-bit analog input
M706->Y:$C018,0,24,U            ; ENC7 time between counts (SCLK cycles)
M710->X:$C018,10,1              ; ENC7 count-write enable control
M711->X:$C018,11,1              ; EQU7 compare flag latch control
M712->X:$C018,12,1              ; EQU7 compare output enable
M713->X:$C018,13,1              ; EQU7 compare invert enable
M714->X:$C018,14,1              ; AENA7/DIR7 Output
M716->X:$C018,16,1              ; EQU7 compare flag
M717->X:$C018,17,1              ; ENC7 position-captured flag
```

```
M718->X:$C018,18,1          ; ENC7 count-error flag
M719->X:$C018,19,1          ; ENC7 3rd channel input status
M720->X:$C018,20,1          ; HMFL7 input status
M721->X:$C018,21,1          ; -LIM7 input status
M722->X:$C018,22,1          ; +LIM7 input status
M723->X:$C018,23,1          ; FAULT7 input status
; Motor #7 Status Bits
M730->Y:$0C94,11,1          ; #7 Stopped-on-position-limit bit
M731->X:$01A5,21,1          ; #7 Positive-end-limit-set bit
M732->X:$01A5,22,1          ; #7 Negative-end-limit-set bit
M733->X:$01A5,13,1          ; #7 Desired-velocity-zero bit
M735->X:$01A5,15,1          ; #7 Dwell-in-progress bit
M737->X:$01A5,17,1          ; #7 Running-program bit
M738->X:$01A5,18,1          ; #7 Open-loop-mode bit
M739->Y:$0C94,14,1          ; #7 Amplifier-enabled status bit
M740->Y:$0C94,0,1           ; #7 In-position bit
M741->Y:$0C94,1,1           ; #7 Warning-following error bit
M742->Y:$0C94,2,1           ; #7 Fatal-following-error bit
M743->Y:$0C94,3,1           ; #7 Amplifier-fault-error bit
M745->Y:$0C94,10,1          ; #7 Home-complete bit
; Motor #7 Move Registers
M761->D:$0190               ; #7 Commanded position (1/[Ix08*32] cts)
M762->D:$0193               ; #7 Actual position (1/[Ix08*32] cts)
M763->D:$0C8B               ; #7 Target (end) position (1/[Ix08*32] cts)
M764->D:$0C93               ; #7 Position bias (1/[Ix08*32] cts)
M765->L:$081C               ; &1 U-axis target position (engineering units)
M766->X:$019B,0,24,S        ; #7 Actual velocity (1/[Ix09*32] cts/cyc)
M767->D:$0195               ; #7 Present master (handwheel) pos (1/[Ix07*32] cts
                            ; of master or (1/[Ix08*32] cts of slaved motor)
M768->X:$01AD,8,16,S        ; #7 Filter Output (DAC bits)
M769->D:$01AE               ; #7 Compensation correction
M770->D:$01A9               ; #7 Present phase position; includes fraction in Y-register
M771->X:$01A9,0,24,S        ; #7 Present phase position (counts*Ix70)
M772->L:$0CAB               ; #7 Variable jog position/distance (counts)
M773->Y:$0C95,0,24,S        ; #7 Encoder home capture offset (counts)
M774->Y:$0CAA,24,S          ; #7 Averaged actual velocity (1/[Ix09*32] cts/cyc)
; Coordinate System &7 Status Bits
M780->X:$0C98,0,1           ; &7 Program-running bit
M781->Y:$0C97,21,1          ; &7 Circle-radius-error bit
M782->Y:$0C97,22,1          ; &7 Run-time-error bit
M784->X:$0C98,4,1           ; &7Continuous motion request
M787->Y:$0C97,17,1          ; &7 In-position bit (AND of motors)
M788->Y:$0C97,18,1          ; &7 Warning-following-error bit (OR)
M789->Y:$0C97,19,1          ; &7 Fatal-following-error bit (OR)
M790->Y:$0C97,20,1          ; &7 Amp-fault-error bit (OR of motors)
; Motor #7 Axis Definition Registers
M791->L:$0CA2               ; #7 X/U/A/B/C-Axis scale factor (cts/unit)
M792->L:$0CA3               ; #7 Y/V-Axis scale factor (cts/unit)
```

```
M793->L:$0CA4                ; #7 Z/W-Axis scale factor (cts/unit)
M794->L:$0CA5                ; #7 Axis offset (cts)
; Coordinate System &7 Variables
M797->X:$0C86,0,24,S         ; &7 Host commanded time base (I10 units)
M798->X:$0C88,0,24,S         ; &7 Present time base (I10 units)
; Registers associated with Encoder/DAC8 (Usually Motor #8)
M801->X:$C01D,0,24,S         ; ENC8 24-bit counter position
M802->Y:$C01A,8,16,S         ; DAC8 16-bit analog output
M803->X:$C01F,0,24,S         ; ENC8 capture/compare position register
M804->X:$0727,0,24,S         ; ENC8 interpolated position (1/32 ct)
M805->Y:$C01F,8,16,S         ; ADC8 16-bit analog input
M806->Y:$C01C,0,24,U         ; ENC8 time between counts (SCLK cycles)
M810->X:$C01C,10,1           ; ENC8 count-write enable control
M811->X:$C01C,11,1           ; EQU8 compare flag latch control
M812->X:$C01C,12,1           ; EQU8 compare output enable
M813->X:$C01C,13,1           ; EQU8 compare invert enable
M814->X:$C01C,14,1           ; AENA8/DIR8 Output
M816->X:$C01C,16,1           ; EQU8 compare flag
M817->X:$C01C,17,1           ; ENC8 position-captured flag
M818->X:$C01C,18,1           ; ENC8 count-error flag
M819->X:$C01C,19,1           ; ENC8 3rd channel input status
M820->X:$C01C,20,1           ; HMFL8 input status
M821->X:$C01C,21,1           ; -LIM8 input status
M822->X:$C01C,22,1           ; +LIM8 input status
M823->X:$C01C,23,1           ; FAULT8 input status
; Motor #8 Status Bits
M830->Y:$0D54,11,1           ; #8 Stopped-on-position-limit bit
M831->X:$01E1,21,1           ; #8 Positive-end-limit-set bit
M832->X:$01E1,22,1           ; #8 Negative-end-limit-set bit
M833->X:$01E1,13,1           ; #8 Desired-velocity-zero bit
M835->X:$01E1,15,1           ; #8 Dwell-in-progress bit
M837->X:$01E1,17,1           ; #8 Running-program bit
M838->X:$01E1,18,1           ; #8 Open-loop-mode bit
M839->Y:$0D54,14,1           ; #8 Amplifier-enabled status bit
M840->Y:$0D54,0,1            ; #8 In-position bit
M841->Y:$0D54,1,1            ; #8 Warning-following error bit
M842->Y:$0D54,2,1            ; #8 Fatal-following-error bit
M843->Y:$0D54,3,1            ; #8 Amplifier-fault-error bit
M845->Y:$0D54,10,1           ; #8 Home-complete bit
; Motor #8 Move Registers
M861->D:$01CC                ; #8 Commanded position (1/[Ix08*32] cts)
M862->D:$01CF                ; #8 Actual position (1/[Ix08*32] cts)
M863->D:$0D4B                ; #8 Target (end) position (1/[Ix08*32] cts)
M864->D:$0D53                ; #8 Position bias (1/[Ix08*32] cts)
M865->L:$081D                ; &1 V-axis target position (engineering units)
M866->X:$01D7,0,24,S         ; #8 Actual velocity (1/[Ix09*32] cts/cyc)
M867->D:$01D1                ; #8 Present master (handwheel) pos (1/[Ix07*32] cts
                             ; of master or (1/[Ix08*32] cts of slaved motor)
```

```
M868->X:$01E9,8,16,S          ; #8 Filter Output (DAC bits)
M869->D:$01EA                 ; #8 Compensation correction
M870->D:$01E5                 ; #8 Present phase position; includes fraction in Y-register
M871->X:$01E5,0,24,S          ; #8 Present phase position (counts*Ix70)
M872->L:$0D6B                 ; #8 Variable jog position/distance (counts)
M873->Y:$0D55,0,24,S          ; #8 Encoder home capture offset (counts)
M874->Y:$0D6A,24,S            ; #8 Averaged actual velocity (1/[Ix09*32] cts/cyc)
; Coordinate System &8 Status Bits
M880->X:$0D58,0,1             ; &8 Program-running bit
M881->Y:$0D57,21,1            ; &8 Circle-radius-error bit
M882->Y:$0D57,22,1            ; &8 Run-time-error bit
M884->X:$0D58,4,1             ; &8 Continuous motion request
M887->Y:$0D57,17,1            ; &8 In-position bit (AND of motors)
M888->Y:$0D57,18,1            ; &8 Warning-following-error bit (OR)
M889->Y:$0D57,19,1            ; &8 Fatal-following-error bit (OR)
M890->Y:$0D57,20,1            ; &8 Amp-fault-error bit (OR of motors)
; Motor #8 Axis Definition Registers
M891->L:$0D62                 ; #8 X/U/A/B/C-Axis scale factor (cts/unit)
M892->L:$0D63                 ; #8 Y/V-Axis scale factor (cts/unit)
M893->L:$0D64                 ; #8 Z/W-Axis scale factor (cts/unit)
M894->L:$0D65                 ; #8 Axis offset (cts)
; Coordinate System &8 Variables
M897->X:$0D46,0,24,S          ; &8 Host commanded time base (I10 units)
M898->X:$0D48,0,24,S          ; &8 Present time base (I10 units)
; Accessory 14 I/O M-Variables (1st ACC-14)
M900->Y:$FFD0,0,1             ; MI/O0
M901->Y:$FFD0,1,1             ; MI/O1
M902->Y:$FFD0,2,1             ; MI/O2
M903->Y:$FFD0,3,1             ; MI/O3
M904->Y:$FFD0,4,1             ; MI/O4
M905->Y:$FFD0,5,1             ; MI/O5
M906->Y:$FFD0,6,1             ; MI/O6
M907->Y:$FFD0,7,1             ; MI/O7
M908->Y:$FFD0,8,1             ; MI/O8
M909->Y:$FFD0,9,1             ; MI/O9
M910->Y:$FFD0,10,1            ; MI/O10
M911->Y:$FFD0,11,1            ; MI/O11
M912->Y:$FFD0,12,1            ; MI/O12
M913->Y:$FFD0,13,1            ; MI/O13
M914->Y:$FFD0,14,1            ; MI/O14
M915->Y:$FFD0,15,1            ; MI/O15
M916->Y:$FFD0,16,1            ; MI/O16
M917->Y:$FFD0,17,1            ; MI/O17
M918->Y:$FFD0,18,1            ; MI/O18
M919->Y:$FFD0,19,1            ; MI/O19
M920->Y:$FFD0,20,1            ; MI/O20
M921->Y:$FFD0,21,1            ; MI/O21
M922->Y:$FFD0,22,1            ; MI/O22
```

```
M923->Y:$FFD0,23,1          ; MI/O23
M924->Y:$FFD1,0,1           ; MI/O24
M925->Y:$FFD1,1,1           ; MI/O25
M926->Y:$FFD1,2,1           ; MI/O26
M927->Y:$FFD1,3,1           ; MI/O27
M928->Y:$FFD1,4,1           ; MI/O28
M929->Y:$FFD1,5,1           ; MI/O29
M930->Y:$FFD1,6,1           ; MI/O30
M931->Y:$FFD1,7,1           ; MI/O31
M932->Y:$FFD1,8,1           ; MI/O32
M933->Y:$FFD1,9,1           ; MI/O33
M934->Y:$FFD1,10,1          ; MI/O34
M935->Y:$FFD1,11,1          ; MI/O35
M936->Y:$FFD1,12,1          ; MI/O36
M937->Y:$FFD1,13,1          ; MI/O37
M938->Y:$FFD1,14,1          ; MI/O38
M939->Y:$FFD1,15,1          ; MI/O39
M940->Y:$FFD1,16,1          ; MI/O40
M941->Y:$FFD1,17,1          ; MI/O41
M942->Y:$FFD1,18,1          ; MI/O42
M943->Y:$FFD1,19,1          ; MI/O43
M944->Y:$FFD1,20,1          ; MI/O44
M945->Y:$FFD1,21,1          ; MI/O45
M946->Y:$FFD1,22,1          ; MI/O46
M947->Y:$FFD1,23,1          ; MI/O47
```

PMAC2 SUGGESTED M-VARIABLE DEFINITIONS

; This file contains suggested definitions for M-variables on the PMAC2. It is similar to the file for the PMAC(1) family of boards, but there are significant differences in the input/output definitions, both for servo registers and general-purpose I/O. Note that these are only suggestions; the user is free to make whatever definitions are desired.

```
; Clear existing definitions
CLOSE                       ; Make sure no buffer is open on PMAC2
M0..1023->*                 ; All M-variables are now self-referenced

; JI/O Port M-variables
M0->Y:$C080,0               ; I/O00 Data Line; J3 Pin 1
M1->Y:$C080,1               ; I/O01 Data Line; J3 Pin 2
M2->Y:$C080,2               ; I/O02 Data Line; J3 Pin 3
M3->Y:$C080,3               ; I/O03 Data Line; J3 Pin 4
M4->Y:$C080,4               ; I/O04 Data Line; J3 Pin 5
M5->Y:$C080,5               ; I/O05 Data Line; J3 Pin 6
M6->Y:$C080,6               ; I/O06 Data Line; J3 Pin 7
M7->Y:$C080,7               ; I/O07 Data Line; J3 Pin 8
M8->Y:$C080,8               ; I/O08 Data Line; J3 Pin 9
M9->Y:$C080,9               ; I/O09 Data Line; J3 Pin 10
M10->Y:$C080,10             ; I/O10 Data Line; J3 Pin 11
M11->Y:$C080,11             ; I/O11 Data Line; J3 Pin 12
M12->Y:$C080,12             ; I/O12 Data Line; J3 Pin 13
M13->Y:$C080,13             ; I/O13 Data Line; J3 Pin 14
M14->Y:$C080,14             ; I/O14 Data Line; J3 Pin 15
M15->Y:$C080,15             ; I/O15 Data Line; J3 Pin 16
M16->Y:$C080,16             ; I/O16 Data Line; J3 Pin 17
M17->Y:$C080,17             ; I/O17 Data Line; J3 Pin 18
M18->Y:$C080,18             ; I/O18 Data Line; J3 Pin 19
M19->Y:$C080,19             ; I/O19 Data Line; J3 Pin 20
M20->Y:$C080,20             ; I/O20 Data Line; J3 Pin 21
M21->Y:$C080,21             ; I/O21 Data Line; J3 Pin 22
M22->Y:$C080,22             ; I/O22 Data Line; J3 Pin 23
M23->Y:$C080,23             ; I/O23 Data Line; J3 Pin 24
M24->Y:$C081,0              ; I/O24 Data Line; J3 Pin 25
M25->Y:$C081,1              ; I/O25 Data Line; J3 Pin 26
M26->Y:$C081,2              ; I/O26 Data Line; J3 Pin 27
M27->Y:$C081,3              ; I/O27 Data Line; J3 Pin 28
M28->Y:$C081,4              ; I/O28 Data Line; J3 Pin 29
M29->Y:$C081,5              ; I/O29 Data Line; J3 Pin 30
M30->Y:$C081,6              ; I/O30 Data Line; J3 Pin 31
M31->Y:$C081,7              ; I/O31 Data Line; J3 Pin 32

M32->X:$C080,0,8            ; Direction control for I/O00 to I/O07
M33->Y:$E800,0              ; Buffer direction control for I/O00 to I/O07
M34->X:$C080,8,8            ; Direction control for I/O08 to I/O15
M35->Y:$E800,1              ; Buffer direction control for I/O08 to I/O15
M36->X:$C080,16,8           ; Direction control for I/O16 to I/O23
```

```
M37->Y:$E800,2                ; Buffer direction control for I/O16 to I/O23
M38->X:$C081,0,8              ; Direction control for I/O24 to I/O31
M39->Y:$E800,3                ; Buffer direction control for I/O24 to I/O31

; JTHW Thumbwheel Multiplexer Port M-variables
M40->Y:$C082,8                ; SEL0 Line; J2 Pin 4
M41->Y:$C082,9                ; SEL1 Line; J2 Pin 6
M42->Y:$C082,10               ; SEL2 Line; J2 Pin 8
M43->Y:$C082,11               ; SEL3 Line; J2 Pin 10
M44->Y:$C082,12               ; SEL4 Line; J2 Pin 12
M45->Y:$C082,13               ; SEL5 Line; J2 Pin 14
M46->Y:$C082,14               ; SEL6 Line; J2 Pin 16
M47->Y:$C082,15               ; SEL7 Line; J2 Pin 18
M48->Y:$C082,8,8,U            ; SEL0-7 Lines treated as a byte
M50->Y:$C082,0                ; DAT0 Line; J2 Pin 3
M51->Y:$C082,1                ; DAT1 Line; J2 Pin 5
M52->Y:$C082,2                ; DAT2 Line; J2 Pin 7
M53->Y:$C082,3                ; DAT3 Line; J2 Pin 9
M54->Y:$C082,4                ; DAT4 Line; J2 Pin 11
M55->Y:$C082,5                ; DAT5 Line; J2 Pin 13
M56->Y:$C082,6                ; DAT6 Line; J2 Pin 15
M57->Y:$C082,7                ; DAT7 Line; J2 Pin 17
M58->Y:$C082,0,8,U            ; DAT0-7 Lines treated as a byte
M60->X:$C082,0,8              ; Direction control for DAT0 to DAT7
M61->Y:$E800,4                ; Buffer direction control for DAT0 to DAT7, PCbus
;M61->Y:$E802,0               ; Buffer direction control for DAT0 to DAT7, VMEbus
M62->X:$C080,8,8              ; Direction control for SEL0 to SEL7
M63->Y:$E800,5                ; Buffer direction control for SEL0 to SEL7, PCbus
;M63->Y:$E802,1               ; Buffer direction control for SEL0 to SEL7, VMEbus

; User timer registers -- count down once per servo cycle
M70->Y:$0700,0,24,S           ; 24-bit countdown user timer
M71->X:$0700,0,24,S           ; 24-bit countdown user timer
M72->Y:$0701,0,24,S           ; 24-bit countdown user timer
M73->X:$0701,0,24,S           ; 24-bit countdown user timer

; Servo cycle counter (read only) -- counts up once per servo cycle
M100->X:$0000,0,24,S          ; 24-bit servo cycle counter

; Gate Array Registers for Channel 1
M101->X:$C001,0,24,S          ; ENC1 24-bit counter position
M102->Y:$C002,8,16,S          ; OUT1A command value; DAC or PWM
M103->X:$C003,0,24,S          ; ENC1 captured position
M104->Y:$C003,8,16,S          ; OUT1B command value; DAC or PWM
M105->X:$0710,8,16,S          ; ADC1A input image value
M106->Y:$0710,8,16,S          ; ADC1B input image value
M107->Y:$C004,8,16,S          ; OUT1C command value; PFM or PWM
M108->Y:$C007,0,24,S          ; ENC1 compare A position
M109->X:$C007,0,24,S          ; ENC1 compare B position
M110->X:$C006,0,24,S          ; ENC1 compare autoincrement value
```

```
M111->X:$C005,11          ; ENC1 compare initial state write enable
M112->X:$C005,12          ; ENC1 compare initial state
M114->X:$C005,14          ; AENA1 output status
M115->X:$C000,19          ; USER1 flag input status
M116->X:$C000,9           ; ENC1 compare output value
M117->X:$C000,11          ; ENC1 capture flag
M118->X:$C000,8           ; ENC1 count error flag
M119->X:$C000,14          ; CHC1 input status
M120->X:$C000,16          ; HMFL1 flag input status
M121->X:$C000,17          ; PLIM1 flag input status
M122->X:$C000,18          ; MLIM1 flag input status
M123->X:$C000,15          ; FAULT1 flag input status
M124->X:$C000,20          ; Channel 1 W flag input status
M125->X:$C000,21          ; Channel 1 V flag input status
M126->X:$C000,22          ; Channel 1 U flag input status
M127->X:$C000,23          ; Channel 1 T flag input status
M128->X:$C000,20,4        ; Channel 1 TUVW inputs as 4-bit value
; Motor #1 Status Bits
M130->Y:$0814,11,1        ; #1 Stopped-on-position-limit bit
M131->X:$003D,21,1        ; #1 Positive-end-limit-set bit
M132->X:$003D,22,1        ; #1 Negative-end-limit-set bit
M133->X:$003D,13,1        ; #1 Desired-velocity-zero bit
M135->X:$003D,15,1        ; #1 Dwell-in-progress bit
M137->X:$003D,17,1        ; #1 Running-program bit
M138->X:$003D,18,1        ; #1 Open-loop-mode bit
M139->Y:$0814,14,1        ; #1 Amplifier-enabled status bit
M140->Y:$0814,0,1         ; #1 In-position bit
M141->Y:$0814,1,1         ; #1 Warning-following error bit
M142->Y:$0814,2,1         ; #1 Fatal-following-error bit
M143->Y:$0814,3,1         ; #1 Amplifier-fault-error bit
M145->Y:$0814,10,1        ; #1 Home-complete bit
; Motor #1 Move Registers
M161->D:$0028             ; #1 Commanded position (1/[Ix08*32] cts)
M162->D:$002B             ; #1 Actual position (1/[Ix08*32] cts)
M163->D:$080B             ; #1 Target (end) position (1/[Ix08*32] cts)
M164->D:$0813             ; #1 Position bias (1/[Ix08*32] cts)
M165->L:$081F             ; &1 X-axis target position (engineering units)
M166->X:$0033,0,24,S      ; #1 Actual velocity (1/[Ix09*32] cts/cyc)
M167->D:$002D             ; #1 Present master pos (1/[Ix07*32] cts)
M168->X:$0043,8,16,S      ; #1 Filter Output (DAC bits)
M169->D:$004A             ; #1 Compensation correction (1/[Ix08*32] cts)
M170->D:$0040             ; #1 Present phase position (including fraction)
M171->X:$0040,24,S        ; #1 Present phase position (counts *Ix70)
M172->L:$082B             ; #1 Variable jog position/distance (cts)
M173->Y:$0815,24,S        ; #1 Encoder home capture position (cts)
M174->Y:$082A,24,S        ; #1 Averaged actual velocity (1/[Ix09*32] cts/cyc)
```

```
; Coordinate System &1 Status Bits
M180->X:$0818,0,1          ; &1 Program-running bit
M181->Y:$0817,21,1         ; &1 Circle-radius-error bit
M182->Y:$0817,22,1         ; &1 Run-time-error bit
M184->X:$0818,0,4          ; &1 Continuous motion request
M187->Y:$0817,17,1         ; &1 In-position bit (AND of motors)
M188->Y:$0817,18,1         ; &1 Warning-following-error bit (OR)
M189->Y:$0817,19,1         ; &1 Fatal-following-error bit (OR)
M190->Y:$0817,20,1         ; &1 Amp-fault-error bit (OR of motors)

; Motor #1 Axis Definition Registers
M191->L:$0822              ; #1 X/U/A/B/C-Axis scale factor (cts/unit)
M192->L:$0823              ; #1 Y/V-Axis scale factor (cts/unit)
M193->L:$0824              ; #1 Z/W-Axis scale factor (cts/unit)
M194->L:$0825              ; #1 Axis offset (cts)

; Coordinate System &1 Variables
M197->X:$0806,0,24,S       ; &1 Host commanded time base (I10 units)
M198->X:$0808,0,24,S       ; &1 Present time base (I10 units)

; Gate Array Registers for Channel 2
M201->X:$C009,0,24,S       ; ENC2 24-bit counter position
M202->Y:$C00A,8,16,S       ; OUT2A command value; DAC or PWM
M203->X:$C00B,0,24,S       ; ENC2 captured position
M204->Y:$C00B,8,16,S       ; OUT2B command value; DAC or PWM
M205->X:$0711,8,16,S       ; ADC2A input image value
M206->Y:$0711,8,16,S       ; ADC2B input image value
M207->Y:$C00C,8,16,S       ; OUT2C command value; PFM or PWM
M208->Y:$C00F,0,24,S       ; ENC2 compare A position
M209->X:$C00F,0,24,S       ; ENC2 compare B position
M210->X:$C00E,0,24,S       ; ENC2 compare autoincrement value
M211->X:$C00D,11           ; ENC2 compare initial state write enable
M212->X:$C00D,12           ; ENC2 compare initial state
M214->X:$C00D,14           ; AENA2 output status
M215->X:$C008,19           ; USER2 flag input status
M216->X:$C008,9            ; ENC2 compare output value
M217->X:$C008,11           ; ENC2 capture flag
M218->X:$C008,8            ; ENC2 count error flag
M219->X:$C008,14           ; CHC2 input status
M220->X:$C008,16           ; HMFL2 flag input status
M221->X:$C008,17           ; PLIM2 flag input status
M222->X:$C008,18           ; MLIM2 flag input status
M223->X:$C008,15           ; FAULT2 flag input status
M224->X:$C008,20           ; Channel 2 W flag input status
M225->X:$C008,21           ; Channel 2 V flag input status
M226->X:$C008,22           ; Channel 2 U flag input status
M227->X:$C008,23           ; Channel 2 T flag input status
M228->X:$C008,20,4         ; Channel 2 TUVW inputs as 4-bit value
```

```
; Motor #2 Status Bits
M230->Y:$08D4,11,1          ; #2 Stopped-on-position-limit bit
M231->X:$0079,21,1          ; #2 Positive-end-limit-set bit
M232->X:$0079,22,1          ; #2 Negative-end-limit-set bit
M233->X:$0079,13,1          ; #2 Desired-velocity-zero bit
M235->X:$0079,15,1          ; #2 Dwell-in-progress bit
M237->X:$0079,17,1          ; #2 Running-program bit
M238->X:$0079,18,1          ; #2 Open-loop-mode bit
M239->Y:$08D4,14,1          ; #2 Amplifier-enabled status bit
M240->Y:$08D4,0,1           ; #2 In-position bit
M241->Y:$08D4,1,1           ; #2 Warning-following error bit
M242->Y:$08D4,2,1           ; #2 Fatal-following-error bit
M243->Y:$08D4,3,1           ; #2 Amplifier-fault-error bit
M245->Y:$08D4,10,1          ; #2 Home-complete bit

; Motor #2 Move Registers
M261->D:$0064               ; #2 Commanded position (1/[Ix08*32] cts)
M262->D:$0067               ; #2 Actual position (1/[Ix08*32] cts)
M263->D:$08CB               ; #2 Target (end) position (1/[Ix08*32] cts)
M264->D:$08D3               ; #2 Position bias (1/[Ix08*32] cts)
M265->L:$0820               ; &1 Y-axis target position (engineering units)
M266->X:$006F,0,24,S        ; #2 Actual velocity (1/[Ix09*32] cts/cyc)
M267->D:$0069               ; #2 Present master pos (1/[Ix07*32] cts)
M268->X:$007F,8,16,S        ; #2 Filter Output (DAC bits)
M269->D:$0086               ; #2 Compensation correction (1/[Ix08*32] cts)
M270->D:$007C               ; #2 Present phase position (including fraction)
M271->X:$007C,24,S          ; #2 Present phase position (counts *Ix70)
M272->L:$08EB               ; #2 Variable jog position/distance (cts)
M273->Y:$08D5,24,S          ; #2 Encoder home capture position (cts)
M274->Y:$08EA,24,S          ; #2 Averaged actual velocity (1/[Ix09*32] cts/cyc)

; Coordinate System &2 Status Bits
M280->X:$08D8,0,1           ; &2 Program-running bit
M281->Y:$08D7,21,1          ; &2 Circle-radius-error bit
M282->Y:$08D7,22,1          ; &2 Run-time-error bit
M284->X:$08D8,0,4           ; &2 Continuous motion request
M287->Y:$08D7,17,1          ; &2 In-position bit (AND of motors)
M288->Y:$08D7,18,1          ; &2 Warning-following-error bit (OR)
M289->Y:$08D7,19,1          ; &2 Fatal-following-error bit (OR)
M290->Y:$08D7,20,1          ; &2 Amp-fault-error bit (OR of motors)

; Motor #2 Axis Definition Registers
M291->L:$08E2               ; #2 X/U/A/B/C-Axis scale factor (cts/unit)
M292->L:$08E3               ; #2 Y/V-Axis scale factor (cts/unit)
M293->L:$08E4               ; #2 Z/W-Axis scale factor (cts/unit)
M294->L:$08E5               ; #2 Axis offset (cts)

; Coordinate System &2 Variables
M297->X:$08C6,0,24,S        ; &2 Host commanded time base (I10 units)
M298->X:$08C8,0,24,S        ; &2 Present time base (I10 units)
```

```
; Gate Array Registers for Channel 3
M301->X:$C011,0,24,S        ; ENC3 24-bit counter position
M302->Y:$C012,8,16,S        ; OUT3A command value; DAC or PWM
M303->X:$C013,0,24,S        ; ENC3 captured position
M304->Y:$C013,8,16,S        ; OUT3B command value; DAC or PWM
M305->X:$0712,8,16,S        ; ADC3A input image value
M306->Y:$0712,8,16,S        ; ADC3B input image value
M307->Y:$C014,8,16,S        ; OUT3C command value; PFM or PWM
M308->Y:$C017,0,24,S        ; ENC3 compare A position
M309->X:$C017,0,24,S        ; ENC3 compare B position
M310->X:$C016,0,24,S        ; ENC3 compare autoincrement value
M311->X:$C015,11            ; ENC3 compare initial state write enable
M312->X:$C015,12            ; ENC3 compare initial state
M314->X:$C015,14            ; AENA3 output status
M315->X:$C010,19            ; USER3 flag input status
M316->X:$C010,9             ; ENC3 compare output value
M317->X:$C010,11            ; ENC3 capture flag
M318->X:$C010,8             ; ENC3 count error flag
M319->X:$C010,14            ; CHC3 input status
M320->X:$C010,16            ; HMFL3 flag input status
M321->X:$C010,17            ; PLIM3 flag input status
M322->X:$C010,18            ; MLIM3 flag input status
M323->X:$C010,15            ; FAULT3 flag input status
M324->X:$C010,20            ; Channel 3 W flag input status
M325->X:$C010,21            ; Channel 3 V flag input status
M326->X:$C010,22            ; Channel 3 U flag input status
M327->X:$C010,23            ; Channel 3 T flag input status
M328->X:$C010,20,4          ; Channel 3 TUVW inputs as 4-bit value

; Motor #3 Status Bits
M330->Y:$0994,11,1          ; #3 Stopped-on-position-limit bit
M331->X:$00B5,21,1          ; #3 Positive-end-limit-set bit
M332->X:$00B5,22,1          ; #3 Negative-end-limit-set bit
M333->X:$00B5,13,1          ; #3 Desired-velocity-zero bit
M335->X:$00B5,15,1          ; #3 Dwell-in-progress bit
M337->X:$00B5,17,1          ; #3 Running-program bit
M338->X:$00B5,18,1          ; #3 Open-loop-mode bit
M339->Y:$0994,14,1          ; #3 Amplifier-enabled status bit
M340->Y:$0994,0,1           ; #3 In-position bit
M341->Y:$0994,1,1           ; #3 Warning-following error bit
M342->Y:$0994,2,1           ; #3 Fatal-following-error bit
M343->Y:$0994,3,1           ; #3 Amplifier-fault-error bit
M345->Y:$0994,10,1          ; #3 Home-complete bit

; Motor #3 Move Registers
M361->D:$00A0               ; #3 Commanded position (1/[Ix08*32] cts)
M362->D:$00A3               ; #3 Actual position (1/[Ix08*32] cts)
M363->D:$098B               ; #3 Target (end) position (1/[Ix08*32] cts)
M364->D:$0993               ; #3 Position bias (1/[Ix08*32] cts)
```

```
M365->L:$0821                ; &1 Z-axis target position (engineering units)
M366->X:$00AB,0,24,S         ; #3 Actual velocity (1/[Ix09*32] cts/cyc)
M367->D:$00A5                ; #3 Present master pos (1/[Ix07*32] cts)
M368->X:$00BB,8,16,S         ; #3 Filter Output (DAC bits)
M369->X:$00C2                ; #3 Compensation correction (1/[Ix08*32] cts)
M370->D:$00B8                ; #3 Present phase position (including fraction)
M371->X:$00B8,24,S           ; #3 Present phase position (counts*Ix70)
M372->L:$09AB                ; #3 Variable jog position/distance (cts)
M373->Y:$0995,24,S           ; #3 Encoder home capture position (cts)
M374->Y:$09AA,24,S           ; #3 Averaged actual velocity (1/[Ix09*32] cts/cyc)

; Coordinate System &3 Status Bits
M380->X:$0998,0,1            ; &3 Program-running bit
M381->Y:$0997,21,1           ; &3 Circle-radius-error bit
M382->Y:$0997,22,1           ; &3 Run-time-error bit
M384->X:$0998,0,4            ; &3 Continuous motion request
M387->Y:$0997,17,1           ; &3 In-position bit (AND of motors)
M388->Y:$0997,18,1           ; &3 Warning-following-error bit (OR)
M389->Y:$0997,19,1           ; &3 Fatal-following-error bit (OR)
M390->Y:$0997,20,1           ; &3 Amp-fault-error bit (OR of motors)

; Motor #3 Axis Definition Registers
M391->L:$09A2                ; #3 X/U/A/B/C-Axis scale factor (cts/unit)
M392->L:$09A3                ; #3 Y/V-Axis scale factor (cts/unit)
M393->L:$09A4                ; #3 Z/W-Axis scale factor (cts/unit)
M394->L:$09A5                ; #3 Axis offset (cts)

; Coordinate System &3 Variables
M397->X:$0986,0,24,S         ; &3 Host commanded time base (I10 units)
M398->X:$0988,0,24,S         ; &3 Present time base (I10 units)

; Gate Array Registers for Channel 4
M401->X:$C019,0,24,S         ; ENC4 24-bit counter position
M402->Y:$C01A,8,16,S         ; OUT4A command value; DAC or PWM
M403->X:$C01B,0,24,S         ; ENC4 captured position
M404->Y:$C01B,8,16,S         ; OUT4B command value; DAC or PWM
M405->X:$0713,8,16,S         ; ADC4A input image value
M406->Y:$0713,8,16,S         ; ADC4B input image value
M407->Y:$C01C,8,16,S         ; OUT4C command value; PFM or PWM
M408->Y:$C01F,0,24,S         ; ENC4 compare A position
M409->X:$C01F,0,24,S         ; ENC4 compare B position
M410->X:$C01E,0,24,S         ; ENC4 compare autoincrement value
M411->X:$C01D,11             ; ENC4 compare initial state write enable
M412->X:$C01D,12             ; ENC4 compare initial state
M414->X:$C01D,14             ; AENA4 output status
M415->X:$C018,19             ; USER4 flag input status
M416->X:$C018,9              ; ENC4 compare output value
M417->X:$C018,11             ; ENC4 capture flag
M418->X:$C018,8              ; ENC4 count error flag
M419->X:$C018,14             ; HMFL4 flag input status
```

```
M420->X:$C018,16          ; CHC4 input status
M421->X:$C018,17          ; PLIM4 flag input status
M422->X:$C018,18          ; MLIM4 flag input status
M423->X:$C018,15          ; FAULT4 flag input status
M424->X:$C018,20          ; Channel 4 W flag input status
M425->X:$C018,21          ; Channel 4 V flag input status
M426->X:$C018,22          ; Channel 4 U flag input status
M427->X:$C018,23          ; Channel 4 T flag input status
M428->X:$C018,20,4        ; Channel 4 TUVW inputs as 4-bit value

; Motor #4 Status Bits
M430->Y:$0A54,11,1        ; #4 Stopped-on-position-limit bit
M431->X:$00F1,21,1        ; #4 Positive-end-limit-set bit
M432->X:$00F1,22,1        ; #4 Negative-end-limit-set bit
M433->X:$00F1,13,1        ; #4 Desired-velocity-zero bit
M435->X:$00F1,15,1        ; #4 Dwell-in-progress bit
M437->X:$00F1,17,1        ; #4 Running-program bit
M438->X:$00F1,18,1        ; #4 Open-loop-mode bit
M439->Y:$0A54,14,1        ; #4 Amplifier-enabled status bit
M440->Y:$0A54,0,1         ; #4 In-position bit
M441->Y:$0A54,1,1         ; #4 Warning-following error bit
M442->Y:$0A54,2,1         ; #4 Fatal-following-error bit
M443->Y:$0A54,3,1         ; #4 Amplifier-fault-error bit
M445->Y:$0A54,10,1        ; #4 Home-complete bit

; Motor #4 Move Registers
M461->D:$00DC             ; #4 Commanded position (1/[Ix08*32] cts)
M462->D:$00DF             ; #4 Actual position (1/[Ix08*32] cts)
M463->D:$0A4B             ; #4 Target (end) position (1/[Ix08*32] cts)
M464->D:$0A53             ; #4 Position bias (1/[Ix08*32] cts)
M465->L:$0819             ; &1 A-axis target position (engineering units)
M466->X:$00E7,0,24,S      ; #4 Actual velocity (1/[Ix09*32] cts/cyc)
M467->D:$00E1             ; #4 Present master pos (1/[Ix07*32] cts)
M468->X:$00F7,8,16,S      ; #4 Filter Output (DAC bits)
M469->D:$00FE             ; #4 Compensation correction (1/[Ix08*32] cts)
M470->D:$00F4             ; #4 Present phase position (including fraction)
M471->X:$00F4,24,S        ; #4 Present phase position (counts*Ix70)
M472->L:$0A6B             ; #4 Variable jog position/distance (cts)
M473->Y:$0A55,24,S        ; #4 Encoder home capture position (cts)
M474->Y:$0A6A,24,S        ; #4 Averaged actual velocity (1/[Ix09*32] cts/cyc)

; Coordinate System &4 Status Bits
M480->X:$0A58,0,1         ; &4 Program-running bit
M481->Y:$0A57,21,1        ; &4 Circle-radius-error bit
M482->Y:$0A57,22,1        ; &4 Run-time-error bit
M484->X:$0A58,0,4         ; &4 Continuous motion request
M487->Y:$0A57,17,1        ; &4 In-position bit (AND of motors)
M488->Y:$0A57,18,1        ; &4 Warning-following-error bit (OR)
M489->Y:$0A57,19,1        ; &4 Fatal-following-error bit (OR)
```

```
M490->Y:$0A57,20,1          ; &4 Amp-fault-error bit (OR of motors)

; Motor #4 Axis Definition Registers
M491->L:$0A62               ; #4 X/U/A/B/C-Axis scale factor (cts/unit)
M492->L:$0A63               ; #4 Y/V-Axis scale factor (cts/unit)
M493->L:$0A64               ; #4 Z/W-Axis scale factor (cts/unit)
M494->L:$0A65               ; #4 Axis offset (cts)

; Coordinate System &4 Variables
M497->X:$0A46,0,24,S        ; &4 Host commanded time base (I10 units)
M498->X:$0A48,0,24,S        ; &4 Present time base (I10 units)

; Gate Array Registers for Channel 5
M501->X:$C021,0,24,S        ; ENC5 24-bit counter position
M502->Y:$C022,8,16,S        ; OUT5A command value; DAC or PWM
M503->X:$C023,0,24,S        ; ENC5 captured position
M504->Y:$C023,8,16,S        ; OUT5B command value; DAC or PWM
M505->X:$0714,8,16,S        ; ADC5A input image value
M506->Y:$0714,8,16,S        ; ADC5B input image value
M507->Y:$C024,8,16,S        ; OUT5C command value; PFM or PWM
M508->Y:$C027,0,24,S        ; ENC5 compare A position
M509->X:$C027,0,24,S        ; ENC5 compare B position
M510->X:$C026,0,24,S        ; ENC5 compare autoincrement value
M511->X:$C025,11            ; ENC5 compare initial state write enable
M512->X:$C025,12            ; ENC5 compare initial state
M514->X:$C025,14            ; AENA5 output status
M515->X:$C020,19            ; USER5 flag input status
M516->X:$C020,9             ; ENC5 compare output value
M517->X:$C020,11            ; ENC5 capture flag
M518->X:$C020,8             ; ENC5 count error flag
M519->X:$C020,14            ; CHC5 input status
M520->X:$C020,16            ; HMFL5 flag input status
M521->X:$C020,17            ; PLIM5 flag input status
M522->X:$C020,18            ; MLIM5 flag input status
M523->X:$C020,15            ; FAULT5 flag input status
M524->X:$C020,20            ; Channel 5 W flag input status
M525->X:$C020,21            ; Channel 5 V flag input status
M526->X:$C020,22            ; Channel 5 U flag input status
M527->X:$C020,23            ; Channel 5 T flag input status
M528->X:$C020,20,4          ; Channel 5 TUVW inputs as 4-bit value

; Motor #5 Status Bits
M530->Y:$0B14,11,1          ; #5 Stopped-on-position-limit bit
M531->X:$012D,21,1          ; #5 Positive-end-limit-set bit
M532->X:$012D,22,1          ; #5 Negative-end-limit-set bit
M533->X:$012D,13,1          ; #5 Desired-velocity-zero bit
M535->X:$012D,15,1          ; #5 Dwell-in-progress bit
M537->X:$012D,17,1          ; #5 Running-program bit
M538->X:$012D,18,1          ; #5 Open-loop-mode bit
M539->Y:$0B14,14,1          ; #5 Amplifier-enabled status bit
```

```
M540->Y:$0B14,0,1          ; #5 In-position bit
M541->Y:$0B14,1,1          ; #5 Warning-following error bit
M542->Y:$0B14,2,1          ; #5 Fatal-following-error bit
M543->Y:$0B14,3,1          ; #5 Amplifier-fault-error bit
M545->Y:$0B14,10,1         ; #5 Home-complete bit

; Motor #5 Move Registers
M561->D:$0118              ; #5 Commanded position (1/[Ix08*32] cts)
M562->D:$011B              ; #5 Actual position (1/[Ix08*32] cts)
M563->D:$0B0B              ; #5 Target (end) position (1/[Ix08*32] cts)
M564->D:$0B13              ; #5 Position bias (1/[Ix08*32] cts)
M565->L:$081A              ; &1 B-axis target position (engineering units)
M566->X:$0123,0,24,S       ; #5 Actual velocity (1/[Ix09*32] cts/cyc)
M567->D:$011D              ; #5 Present master pos (1/[Ix07*32] cts)
M568->X:$0133,8,16,S       ; #5 Filter Output (DAC bits)
M569->D:$013A              ; #5 Compensation correction (1/[Ix08*32] cts)
M570->D:$0130              ; #5 Present phase position (including fraction)
M571->X:$0130,24,S         ; #5 Present phase position (counts*Ix70)
M572->L:$0B2B              ; #5 Variable jog position/distance (cts)
M573->Y:$0B15,24,S         ; #5 Encoder home capture position (cts)
M574->Y:$0B2A,24,S         ; #5 Averaged actual velocity (1/[Ix09*32] cts/cyc)

; Coordinate System &5 Status Bits
M580->X:$0B18,0,1          ; &5 Program-running bit
M581->Y:$0B17,21,1         ; &5 Circle-radius-error bit
M582->Y:$0B17,22,1         ; &5 Run-time-error bit
M584->X:$0B18,0,4          ; &5 Continuous motion request
M587->Y:$0B17,17,1         ; &5 In-position bit (AND of motors)
M588->Y:$0B17,18,1         ; &5 Warning-following-error bit (OR)
M589->Y:$0B17,19,1         ; &5 Fatal-following-error bit (OR)
M590->Y:$0B17,20,1         ; &5 Amp-fault-error bit (OR of motors)

; Motor #5 Axis Definition Registers
M591->L:$0B22              ; #5 X/U/A/B/C-Axis scale factor (cts/unit)
M592->L:$0B23              ; #5 Y/V-Axis scale factor (cts/unit)
M593->L:$0B24              ; #5 Z/W-Axis scale factor (cts/unit)
M594->L:$0B25              ; #5 Axis offset (cts)

; Coordinate System &5 Variables
M597->X:$0B06,0,24,S       ; &5 Host commanded time base (I10 units)
M598->X:$0B08,0,24,S             ; &5 Present time base (I10 units)

; Gate Array Registers for Channel 6
M601->X:$C029,0,24,S       ; ENC6 24-bit counter position
M602->Y:$C02A,8,16,S       ; OUT6A command value; DAC or PWM
M603->X:$C02B,0,24,S       ; ENC6 captured position
M604->Y:$C02B,8,16,S       ; OUT6B command value; DAC or PWM
M605->X:$0715,8,16,S       ; ADC6A input image value
M606->Y:$0715,8,16,S       ; ADC6B input image value
M607->Y:$C02C,8,16,S       ; OUT6C command value; PFM or PWM
```

```
M608->Y:$C02F,0,24,S        ; ENC6 compare A position
M609->X:$C02F,0,24,S        ; ENC6 compare B position
M610->X:$C02E,0,24,S        ; ENC6 compare autoincrement value
M611->X:$C02D,11            ; ENC6 compare initial state write enable
M612->X:$C02D,12            ; ENC6 compare initial state
M614->X:$C02D,14            ; AENA6 output status
M615->X:$C028,19            ; USER6 flag input status
M616->X:$C028,9             ; ENC6 compare output value
M617->X:$C028,11            ; ENC6 capture flag
M618->X:$C028,8             ; ENC6 count error flag
M619->X:$C028,14            ; CHC6 input status
M620->X:$C028,16            ; HMFL6 flag input status
M621->X:$C028,17            ; PLIM6 flag input status
M622->X:$C028,18            ; MLIM6 flag input status
M623->X:$C028,15            ; FAULT6 flag input status
M624->X:$C028,20            ; Channel 6 W flag input status
M625->X:$C028,21            ; Channel 6 V flag input status
M626->X:$C028,22            ; Channel 6 U flag input status
M627->X:$C028,23            ; Channel 6 T flag input status
M628->X:$C028,20,4          ; Channel 6 TUVW inputs as 4-bit value

; Motor #6 Status Bits
M630->Y:$0BD4,11,1          ; #6 Stopped-on-position-limit bit
M631->X:$0169,21,1          ; #6 Positive-end-limit-set bit
M632->X:$0169,22,1          ; #6 Negative-end-limit-set bit
M633->X:$0169,13,1          ; #6 Desired-velocity-zero bit
M635->X:$0169,15,1          ; #6 Dwell-in-progress bit
M637->X:$0169,17,1          ; #6 Running-program bit
M638->X:$0169,18,1          ; #6 Open-loop-mode bit
M639->Y:$0BD4,14,1          ; #6 Amplifier-enabled status bit
M640->Y:$0BD4,0,1           ; #6 In-position bit
M641->Y:$0BD4,1,1           ; #6 Warning-following error bit
M642->Y:$0BD4,2,1           ; #6 Fatal-following-error bit
M643->Y:$0BD4,3,1           ; #6 Amplifier-fault-error bit
M645->Y:$0BD4,10,1          ; #6 Home-complete bit

; Motor #6 Move Registers
M661->D:$0154               ; #6 Commanded position (1/[Ix08*32] cts)
M662->D:$0157               ; #6 Actual position (1/[Ix08*32] cts)
M663->D:$0BCB               ; #6 Target (end) position (1/[Ix08*32] cts)
M664->D:$0BD3               ; #6 Position bias (1/[Ix08*32] cts)
M665->L:$081B               ; &1 C-axis target position (engineering units)
M666->X:$015F,0,24,S        ; #6 Actual velocity (1/[Ix09*32] cts/cyc)
M667->D:$0159               ; #6 Present master pos (1/[Ix07*32] cts)
M668->X:$016F,8,16,S        ; #6 Filter Output (DAC bits)
M669->D:$0176               ; #6 Compensation correction (1/[Ix08*32] cts)
M670->D:$016C               ; #6 Present phase position (including fraction)
M671->X:$016C,24,S          ; #6 Present phase position (counts*Ix70)
```

```
M672->L:$0BEB                ; #6 Variable jog position/distance (cts)
M673->Y:$0BD5,24,S           ; #6 Encoder home capture position (cts)
M674->Y:$0BEA,24,S           ; #6 Averaged actual velocity (1/[Ix09*32] cts/cyc)

; Coordinate System &6 Status Bits
M680->X:$0BD8,0,1            ; &6 Program-running bit
M681->Y:$0BD7,21,1           ; &6 Circle-radius-error bit
M682->Y:$0BD7,22,1           ; &6 Run-time-error bit
M684->X:$0BD8,0,4            ; &6 Continuous motion request
M687->Y:$0BD7,17,1           ; &6 In-position bit (AND of motors)
M688->Y:$0BD7,18,1           ; &6 Warning-following-error bit (OR)
M689->Y:$0BD7,19,1           ; &6 Fatal-following-error bit (OR)
M690->Y:$0BD7,20,1           ; &6 Amp-fault-error bit (OR of motors)

; Motor #6 Axis Definition Registers
M691->L:$0BE2                ; #6 X/U/A/B/C-Axis scale factor (cts/unit)
M692->L:$0BE3                ; #6 Y/V-Axis scale factor (cts/unit)
M693->L:$0BE4                ; #6 Z/W-Axis scale factor (cts/unit)
M694->L:$0BE5                ; #6 Axis offset (cts)

; Coordinate System &6 Variables
M697->X:$0BC6,0,24,S         ; &6 Host commanded time base (I10 units)
M698->X:$0BC8,0,24,S         ; &6 Present time base (I10 units)

; Gate Array Registers for Channel 7
M701->X:$C031,0,24,S         ; ENC7 24-bit counter position
M702->Y:$C032,8,16,S         ; OUT7A command value; DAC or PWM
M703->X:$C033,0,24,S         ; ENC7 captured position
M704->Y:$C033,8,16,S         ; OUT7B command value; DAC or PWM
M705->X:$0716,8,16,S         ; ADC7A input image value
M706->Y:$0716,8,16,S         ; ADC7B input image value
M707->Y:$C034,8,16,S         ; OUT7C command value; PFM or PWM
M708->Y:$C037,0,24,S         ; ENC7 compare A position
M709->X:$C037,0,24,S         ; ENC7 compare B position
M710->X:$C036,0,24,S         ; ENC7 compare autoincrement value
M711->X:$C035,11             ; ENC7 compare initial state write enable
M712->X:$C035,12             ; ENC7 compare initial state
M714->X:$C035,14             ; AENA7 output status
M715->X:$C030,19             ; CHC7 input status
M716->X:$C030,9              ; ENC7 compare output value
M717->X:$C030,11             ; ENC7 capture flag
M718->X:$C030,8              ; ENC7 count error flag
M719->X:$C030,14             ; CHC7 input status
M720->X:$C030,16             ; HMFL7 flag input status
M721->X:$C030,17             ; PLIM7 flag input status
M722->X:$C030,18             ; MLIM7 flag input status
M723->X:$C030,15             ; FAULT7 flag input status
M724->X:$C030,20             ; Channel 7 W flag input status
M725->X:$C030,21             ; Channel 7 V flag input status
M726->X:$C030,22             ; Channel 7 U flag input status
```

```
M727->X:$C030,23          ; Channel 7 T flag input status
M728->X:$C030,20,4        ; Channel 7 TUVW inputs as 4-bit value

; Motor #7 Status Bits
M730->Y:$0C94,11,1        ; #7 Stopped-on-position-limit bit
M731->X:$01A5,21,1        ; #7 Positive-end-limit-set bit
M732->X:$01A5,22,1        ; #7 Negative-end-limit-set bit
M733->X:$01A5,13,1        ; #7 Desired-velocity-zero bit
M735->X:$01A5,15,1        ; #7 Dwell-in-progress bit
M737->X:$01A5,17,1        ; #7 Running-program bit
M738->X:$01A5,18,1        ; #7 Open-loop-mode bit
M739->Y:$0C94,14,1        ; #7 Amplifier-enabled status bit
M740->Y:$0C94,0,1         ; #7 In-position bit
M741->Y:$0C94,1,1         ; #7 Warning-following error bit
M742->Y:$0C94,2,1         ; #7 Fatal-following-error bit
M743->Y:$0C94,3,1         ; #7 Amplifier-fault-error bit
M745->Y:$0C94,10,1        ; #7 Home-complete bit

; Motor #7 Move Registers
M761->D:$0190             ; #7 Commanded position (1/[Ix08*32] cts)
M762->D:$0193             ; #7 Actual position (1/[Ix08*32] cts)
M763->D:$0C8B             ; #7 Target (end) position (1/[Ix08*32] cts)
M764->D:$0C93             ; #7 Position bias (1/[Ix08*32] cts)
M765->L:$081C             ; &1 U-axis target position (engineering units)
M766->X:$019B,0,24,S      ; #7 Actual velocity (1/[Ix09*32] cts/cyc)
M767->D:$0195             ; #7 Present master pos (1/[Ix07*32] cts)
M768->X:$01AB,8,16,S      ; #7 Filter Output (DAC bits)
M769->D:$01B2             ; #7 Compensation correction (1/[Ix08*32] cts)
M770->D:$01A8             ; #7 Present phase position (including fraction)
M771->X:$01A8,24,S        ; #7 Present phase position (counts*Ix70)
M772->L:$0CAB             ; #7 Variable jog position/distance (cts)
M773->Y:$0C95,24,S        ; #7 Encoder home capture position (cts)
M774->Y:$0CAA,24,S        ; #7 Averaged actual velocity (1/[Ix09*32] cts/cyc)

; Coordinate System &7 Status Bits
M780->X:$0C98,0,1         ; &7 Program-running bit
M781->Y:$0C97,21,1        ; &7 Circle-radius-error bit
M782->Y:$0C97,22,1        ; &7 Run-time-error bit
M784->X:$0C98,0,4         ; &7 Continuous motion request
M787->Y:$0C97,17,1        ; &7 In-position bit (AND of motors)
M788->Y:$0C97,18,1        ; &7 Warning-following-error bit (OR)
M789->Y:$0C97,19,1        ; &7 Fatal-following-error bit (OR)
M790->Y:$0C97,20,1        ; &7 Amp-fault-error bit (OR of motors)

; Motor #7 Axis Definition Registers
M791->L:$0CA2             ; #7 X/U/A/B/C-Axis scale factor (cts/unit)
M792->L:$0CA3             ; #7 Y/V-Axis scale factor (cts/unit)
M793->L:$0CA4             ; #7 Z/W-Axis scale factor (cts/unit)
M794->L:$0CA5             ; #7 Axis offset (cts)
```

```
; Coordinate System &7 Variables
M797->X:$0C86,0,24,S        ; &7 Host commanded time base (I10 units)
M798->X:$0C88,0,24,S        ; &7 Present time base (I10 units)

; Gate Array Registers for Channel 8
M801->X:$C039,0,24,S        ; ENC8 24-bit counter position
M802->Y:$C03A,8,16,S        ; OUT8A command value; DAC or PWM
M803->X:$C03B,0,24,S        ; ENC8 captured position
M804->Y:$C03B,8,16,S        ; OUT8B command value; DAC or PWM
M805->X:$0717,8,16,S        ; ADC8A input image value
M806->Y:$0717,8,16,S        ; ADC8B input image value
M807->Y:$C03C,8,16,S        ; OUT8C command value; PFM or PWM
M808->Y:$C03F,0,24,S        ; ENC8 compare A position
M809->X:$C03F,0,24,S        ; ENC8 compare B position
M810->X:$C03E,0,24,S        ; ENC8 compare autoincrement value
M811->X:$C03D,11            ; ENC8 compare initial state write enable
M812->X:$C03D,12            ; ENC8 compare initial state
M814->X:$C03D,14            ; AENA8 output status
M815->X:$C038,19            ; USER8 flag input status
M816->X:$C038,9             ; ENC8 compare output value
M817->X:$C038,11            ; ENC8 capture flag
M818->X:$C038,8             ; ENC8 count error flag
M819->X:$C038,14            ; CHC8 input status
M820->X:$C038,16            ; HMFL8 flag input status
M821->X:$C038,17            ; PLIM8 flag input status
M822->X:$C038,18            ; MLIM8 flag input status
M823->X:$C038,15            ; FAULT8 flag input status
M824->X:$C038,20            ; Channel 8 W flag input status
M825->X:$C038,21            ; Channel 8 V flag input status
M826->X:$C038,22            ; Channel 8 U flag input status
M827->X:$C038,23            ; Channel 8 T flag input status
M828->X:$C038,20,4          ; Channel 8 TUVW inputs as 4-bit value

; Motor #8 Status Bits
M830->Y:$0D54,11,1          ; #8 Stopped-on-position-limit bit
M831->X:$01E1,21,1          ; #8 Positive-end-limit-set bit
M832->X:$01E1,22,1          ; #8 Negative-end-limit-set bit
M833->X:$01E1,13,1          ; #8 Desired-velocity-zero bit
M835->X:$01E1,15,1          ; #8 Dwell-in-progress bit
M837->X:$01E1,17,1          ; #8 Running-program bit
M838->X:$01E1,18,1          ; #8 Open-loop-mode bit
M839->Y:$0D54,14,1          ; #8 Amplifier-enabled status bit
M840->Y:$0D54,0,1           ; #8 In-position bit
M841->Y:$0D54,1,1           ; #8 Warning-following error bit
M842->Y:$0D54,2,1           ; #8 Fatal-following-error bit
M843->Y:$0D54,3,1           ; #8 Amplifier-fault-error bit
M845->Y:$0D54,10,1          ; #8 Home-complete bit
```

```
; Motor #8 Move Registers
M861->D:$01CC                ; #8 Commanded position (1/[Ix08*32] cts)
M862->D:$01CF                ; #8 Actual position (1/[Ix08*32] cts)
M863->D:$0D4B                ; #8 Target (end) position (1/[Ix08*32] cts)
M864->D:$0D53                ; #8 Position bias (1/[Ix08*32] cts)
M865->L:$081D                ; &1 V-axis target position (engineering units)
M866->X:$01D7,0,24,S         ; #8 Actual velocity (1/[Ix09*32] cts/cyc)
M867->D:$01D1                ; #8 Present master pos (1/[Ix07*32] cts)
M868->X:$01E7,8,16,S         ; #8 Filter Output (DAC bits)
M869->D:$01EE                ; #8 Compensation correction (1/[Ix08*32] cts)
M870->D:$01E4                ; #8 Present phase position (including fraction)
M871->X:$01E4,24,S           ; #8 Present phase position (counts*Ix70)
M872->L:$0D6B                ; #8 Variable jog position/distance (cts)
M873->Y:$0D55,24,S           ; #8 Encoder home capture position (cts)
M874->Y:$0D6A,24,S           ; #8 Averaged actual velocity (1/[Ix09*32] cts/cyc)

; Coordinate System &8 Status Bits
M880->X:$0D58,0,1            ; &8 Program-running bit
M881->Y:$0D57,21,1           ; &8 Circle-radius-error bit
M882->Y:$0D57,22,1           ; &8 Run-time-error bit
M884->X:$0D58,0,4            ; &8 Continuous motion request
M887->Y:$0D57,17,1           ; &8 In-position bit (AND of motors)
M888->Y:$0D57,18,1           ; &8 Warning-following-error bit (OR)
M889->Y:$0D57,19,1           ; &8 Fatal-following-error bit (OR)
M890->Y:$0D57,20,1           ; &8 Amp-fault-error bit (OR of motors)

; Motor #8 Axis Definition Registers
M891->L:$0D62                ; #8 X/U/A/B/C-Axis scale factor (cts/unit)
M892->L:$0D63                ; #8 Y/V-Axis scale factor (cts/unit)
M893->L:$0D64                ; #8 Z/W-Axis scale factor (cts/unit)
M894->L:$0D65                ; #8 Axis offset (cts)

; Coordinate System &8 Variables
M897->X:$0D46,0,24,S         ; &8 Host commanded time base (I10 units)
M898->X:$0D48,0,24,S         ; &8 Present time base (I10 units)

; Accessory 14 I/O M-Variables (1st ACC-14)
M900->Y:$FFD0,0,1            ; MI/O0
M901->Y:$FFD0,1,1            ; MI/O1
M902->Y:$FFD0,2,1            ; MI/O2
M903->Y:$FFD0,3,1            ; MI/O3
M904->Y:$FFD0,4,1            ; MI/O4
M905->Y:$FFD0,5,1            ; MI/O5
M906->Y:$FFD0,6,1            ; MI/O6
M907->Y:$FFD0,7,1            ; MI/O7
M908->Y:$FFD0,8,1            ; MI/O8
M909->Y:$FFD0,9,1            ; MI/O9
M910->Y:$FFD0,10,1           ; MI/O10
M911->Y:$FFD0,11,1           ; MI/O11
M912->Y:$FFD0,12,1           ; MI/O12
```

```
M913->Y:$FFD0,13,1          ; MI/O13
M914->Y:$FFD0,14,1          ; MI/O14
M915->Y:$FFD0,15,1          ; MI/O15
M916->Y:$FFD0,16,1          ; MI/O16
M917->Y:$FFD0,17,1          ; MI/O17
M918->Y:$FFD0,18,1          ; MI/O18
M919->Y:$FFD0,19,1          ; MI/O19
M920->Y:$FFD0,20,1          ; MI/O20
M921->Y:$FFD0,21,1          ; MI/O21
M922->Y:$FFD0,22,1          ; MI/O22
M923->Y:$FFD0,23,1          ; MI/O23
M924->Y:$FFD1,0,1           ; MI/O24
M925->Y:$FFD1,1,1           ; MI/O25
M926->Y:$FFD1,2,1           ; MI/O26
M927->Y:$FFD1,3,1           ; MI/O27
M928->Y:$FFD1,4,1           ; MI/O28
M929->Y:$FFD1,5,1           ; MI/O29
M930->Y:$FFD1,6,1           ; MI/O30
M931->Y:$FFD1,7,1           ; MI/O31
M932->Y:$FFD1,8,1           ; MI/O32
M933->Y:$FFD1,9,1           ; MI/O33
M934->Y:$FFD1,10,1          ; MI/O34
M935->Y:$FFD1,11,1          ; MI/O35
M936->Y:$FFD1,12,1          ; MI/O36
M937->Y:$FFD1,13,1          ; MI/O37
M938->Y:$FFD1,14,1          ; MI/O38
M939->Y:$FFD1,15,1          ; MI/O39
M940->Y:$FFD1,16,1          ; MI/O40
M941->Y:$FFD1,17,1          ; MI/O41
M942->Y:$FFD1,18,1          ; MI/O42
M943->Y:$FFD1,19,1          ; MI/O43
M944->Y:$FFD1,20,1          ; MI/O44
M945->Y:$FFD1,21,1          ; MI/O45
M946->Y:$FFD1,22,1          ; MI/O46
M947->Y:$FFD1,23,1          ; MI/O47

; JANA Analog Input Port M-Variables
M990->X:$0708,0,24,U        ; 1st CONFIG_W1 and CONFIG_W2
M991->X:$0709,0,24,U        ; 2nd CONFIG_W1 and CONFIG_W2
M992->X:$070A,0,24,U        ; 3rd CONFIG_W1 and CONFIG_W2
M993->X:$070B,0,24,U        ; 4th CONFIG_W1 and CONFIG_W2
M994->X:$070C,0,24,U        ; 5th CONFIG_W1 and CONFIG_W2
M995->X:$070D,0,24,U        ; 6th CONFIG_W1 and CONFIG_W2
M996->X:$070E,0,24,U        ; 7th CONFIG_W1 and CONFIG_W2
M997->X:$070F,0,24,U        ; 8th CONFIG_W1 and CONFIG_W2
M1000->Y:$0708,0,12,{f}     ; ANAI00 image register; from J1 pin 1
M1001->Y:$0709,0,12,{f}     ; ANAI01 image register; from J1 pin 2
M1002->Y:$070A,0,12,{f}     ; ANAI02 image register; from J1 pin 3
```

```
M1003->Y:$070B,0,12,{f}      ; ANAI03 image register; from J1 pin 4
M1004->Y:$070C,0,12,{f}      ; ANAI04 image register; from J1 pin 5
M1005->Y:$070D,0,12,{f}      ; ANAI05 image register; from J1 pin 6
M1006->Y:$070E,0,12,{f}      ; ANAI06 image register; from J1 pin 7
M1007->Y:$070F,0,12,{f}      ; ANAI07 image register; from J1 pin 8
M1008->Y:$0708,12,12,{f}     ; ANAI08 image register; from J1 pin 9
M1009->Y:$0709,12,12,{f}     ; ANAI09 image register; from J1 pin 10
M1010->Y:$070A,12,12,{f}     ; ANAI10 image register; from J1 pin 11
M1011->Y:$070B,12,12,{f}     ; ANAI11 image register; from J1 pin 12
M1012->Y:$070C,12,12,{f}     ; ANAI12 image register; from J1 pin 13
M1013->Y:$070D,12,12,{f}     ; ANAI13 image register; from J1 pin 14
M1014->Y:$070E,12,12,{f}     ; ANAI14 image register; from J1 pin 15
M1015->Y:$070F,12,12,{f}     ; ANAI15 image register; from J1 pin 16

; where {f} should be a U if the channel is read as an unsigned quantity,
; or an S if the channel is read as a signed quantity.
```

PMAC FIRMWARE UPDATES

To update to a new revision of firmware:

Battery-backed PMAC(1) boards

For a PMAC(1) controller with battery-backed main memory, the firmware is located in a PROM IC that cannot be written to in the field. For these controllers, the existing PROM IC must be removed from its socket on the board, and a new PROM IC installed in its place.

The PROM IC for each of these controllers is:

PMAC(1)-PC: CPU-board IC U5
PMAC(1)-VME: CPU-board IC U5
PMAC(1)-Lite: U5

Flash-backed PMAC(1), all PMAC2 boards

For a PMAC(1) board with flash backed memory or any PMAC2 board, the firmware is located in a flash-memory IC that can be written to in the field. PMAC(1) controllers have firmware in flash memory if they are ordered with Option 4A, or any Option 5x. PMAC1.5-Lite, Mini-PMAC(1), and PMAC1.5-STD controllers always have firmware in flash memory.

For these controllers, the firmware can be updated by installing the "bootstrap" jumper before powering the board up. When the board is powered up and communications is established with the board by the PMAC Executive program, the program will notice that the board is in "bootstrap mode" and ask you if you want to load new firmware. Just follow its directions from this point. Remember to remove the bootstrap jumper before the next time you power up or reset the board.

The bootstrap jumper for each of these controllers is:
PMAC(1)-PC: CPU-board jumper E4, E7*
PMAC(1)-VME: CPU-board jumper E4, E7*
PMAC(1)-PCI: CPU-board jumper E4, E7*
PMAC1.5-Lite: E106 (connect pins 2 and 3)
Mini-PMAC(1): E51
PMAC1.5-STD: E51
PMAC2-PC: CPU-board jumper E4, E7*
PMAC2-VME: CPU-board jumper E4, E7*
PMAC2-PCI: CPU-board jumper E4, E7*
PMAC2-Lite: E0 (connect pins 2 and 3)
Mini-PMAC2: E0
PMAC2-PC Ultralite: E0
PMAC2-VME Ultralite: E3

*E4 on Universal CPU board (602705); E7 on "Flex" CPU board (603605)

Update Summary: From V1.15 to V1.16 (July 1996)

Changes

1. With I99=0 backlash hysteresis is 0 counts, not 4 counts.
2. With I89=0 cutter comp outside corners only add arc if change in directed angle is greater than 90^{o}; formerly 1^{o}.
3. Default changed from 10 to 0.
4. Default changed from 0 to 37137 (1 second slew by default).
5. Full circles executed on any arc command (with IJK center specification) smaller than 2^{-20} part of circle (0.5 arc second); formerly 2^{-34} part of circle.
6. Boundary of cutter compensation lead-in and lead-out moves on inside corners changed slightly; now offset directly from uncompensated destination point perpendicular to fully compensated move only.

Additions

Features

1. User-written phase commutation routines (Ix59).
2. "Stepper-motor" phasing search method for synchronous motors (Ix80).
3. Hall-Effect power-on phase read (Ix81).
4. Integrated current (I^2T) limiting protection (Ix57, Ix58).
5. Integrated position error limiting protection (Ix63).
6. Programmable backlash hysteresis (I99).
7. Backlash compensation tables (DEFINE BLCOMP).
8. Torque compensation tables (DEFINE TCOMP).
9. MACRO ring support enhancements.
10. Teach/Learn functionality (LEARN).
11. Non-uniform spline mode (SPLINE2).
12. Variable jog commands (J=*, J:*, J^*).
13. Jog-until-trigger commands ({jog}^{constant}).
14. Program move-until-trigger commands ({axis}{data}^{data}).
15. "Torque-limiting" triggering (trigger on warning following error) (Ix03).
16. Extended G, M, T, and D code ranges.
17. Better control of outside corners with cutter compensation (I89).
18. PRELUDE functions in motion programs.
19. Ability to suppress carriage return after SEND and CMD responses (I62).
20. Unsigned A/D feedback ($18, $58 conversions).
21. Parallel feedback in high 16 bits ($2C, $3C, $6C, $7C conversions).
22. Support for ladder-logic compiler in special firmware version.

I-Variables

1. **I55** -- DPRAM Background Variable Buffer Control (V1.15G).
2. **I62** -- Internal Message Carriage Return Control.
3. **I89** -- Cutter Comp Outside Angle Break Point.
4. **I99** -- Backlash Hysteresis.
5. **Ix02** -- Range extended: bit 19=1 specifies X-register output.
6. **Ix25** -- Range extended: bit 18=1 specifies MACRO node for flags.
7. **Ix30** -- Range extended: negative values can be used to invert output polarity.
8. **Ix57** -- Continuous Current Limit for I^2T current fault.
9. **Ix58** -- Integrated Current Fault Level for I^2T current fault.
10. **Ix59** -- Range extended to 0-3; values of 2&3 enable user-written commutation.

11. **Ix63** -- Range extended; negative values permit integrated following error fault.
12. **Ix80** -- Range extended to support "stepper" power-on phasing search.
13. **Ix81** -- Range extended to support hall-effect phase read.
14. **Ix83** -- Range extended: bit 19=1 specifies Y-register feedback.
15. **I1000** -- MACRO node auxiliary register enable.
16. **I1001** -- MACRO Ring Check Time Period.
17. **TWS-format M-variables** -- Extended to add parity bits in backward-compatible format; bit 6 of global status word X:$0003 is set to 1 if the most recent TWS read or write operation resulted in a parity error.

Conversion Table Entries

1. **$2C, $3C, $6C, $7C** -- parallel feedback shifted right 3 bits (meant for feedback appearing in high 16 bits); supports rollover.
2. **$18, $58** -- unsigned 16-bit A/D feedback, no rollover of source; to support ACC-28B.

On-Line Commands

1. **DEFINE BLCOMP** -- Establishes backlash table for addressed motor.
2. **DEFINE TCOMP** -- Establishes torque compensation table for addressed Motor.
3. **DELETE BLCOMP** -- Erases backlash table for addressed motor.
4. **DELETE TCOMP** -- Erases torque compensation table for addressed motor.
5. **J=*** -- Variable jog-to-position; destination in L:$082B, etc. (V1.15E).
6. **J:*** -- Variable incremental jog; distance in L:$082B, etc. (V1.15E).
7. **J^*** -- Variable incremental jog, distance in L:$082B, etc. (V1.15E).
8. **J=={constant}** -- Jog to specified position, make that "pre-jog" position.
9. **{jog command}^{constant}** -- Jog until trigger, final value specifies distance from trigger position to stop.
10. **LEARN** -- Reads present commanded positions for all motors in coordinate system, converts to axis positions, adds axis commands to open motion program.
11. **MFLUSH** -- Clears synchronous M-variable stack without executing.
12. **LIST LDS** -- PMAC reports addresses of special compiled PLC ladder-logic routines for cross-compiler (special firmware required).
13. **TYPE** -- PMAC reports hardware and software configuration.

Motion Program Commands

1. **{axis}{data}^{data}** -- Move-until-trigger for RAPID mode moves.
2. **SPLINE2** -- Puts program in non-uniform B-spline mode.
3. **G, M, T, D** codes -- Extended to support range of 0-999.99999; formerly 0-99.99999.
4. **PRELUDE1{command}** -- Adds **{command}** before subsequent move commands.
5. **PRELUDE0** -- turns off prelude function.

DPRAM Structures

Second binary rotary program download buffer added (V1.15G).

Refinements

1. Full circle moves always executed after starting position recalculated due to PMATCH or axis matrix transformation (more tolerance between start and end positions permitted; see Changes, above).
2. DISPLAY of I-variables fixed.
3. Fast versions of PMAC can interface to ACC-8D Opt 9.
4. Fast versions of PMAC can interface to NC control panel and ACC-34C.
5. O-commands now saturate at +/-100% of Ix69.
6. I52 default set to 37137 for default 1 second stop on '\' program hold command.
7. I11 default set to 0 so move sequences start as soon as calculations are finished.
8. Ix77, Ix86 reported as signed values.
9. RAPID declaration alone on a motion program line no longer causes zero-distance move taking 2*TA time.
10. Ix10 absolute servo-position read copies axis definition offset into position bias register.
11. Maximum commanded speed increased by factor of 3 to 768M/Ix08 cts/sec (8M cts/sec at default Ix08 value).
12. Cutter comp lead-in move destination, lead-out move origin, changed slightly.
13. Ix30 now can take negative values, permitting output polarity reversal.
14. (PMAC2) ADC1A,B to ADC8A,B registers in DSPGATE1 copied into RAM registers $0710 - $0717 one pair per phase cycle for reliable servo & program access.
15. **$$$***** command kills all motors before re-initialization.

Update Summary: From V1.16 to V1.16A (Sept 1996)

1. Corrected problems with **PRELUDE** subroutine calls
2. Serial port forces CTS true on power-up/reset except for cards numbered greater than 0 on serial daisychain (I1 = 2 or 3)
3. Cutter compensation outside corner lead-in and lead-out moves always add an arc, regardless of the setting of I89.
4. When Ix83 bit 19 is set to 1 to read Y-register as commutation feedback, register is read just as an X-register would be (all 24 bits). This function is now enabled for PMAC(1).
5. For MACRO interface, if I1000=0, firmware no longer automatically reads I995 register, an action which clears the MACRO error flags.

Update Summary: From V1.16A to V1.16B (Oct 1996)

Corrected STDbus interface problem

Update Summary: From V1.16B to V1.16C (Apr 1997)

1. Corrected problem with interrupt function for PMAC2 DPRAM ASCII buffer.
2. Corrected memory clear problem on **$$$** software reset that could affect program and firmware checksum verification, table-based math function operation.
3. Corrected cutter compensation lead-in/lead-out problem that occurred when lead-in move was collinear with first compensated move, or lead-out move was collinear with last compensated move.
4. Corrected operation of buffered motion command **HOMEZn** for rotary axes.
5. Completed implementation of "Auto-Abort on Run-Time Error" function. Ix97 (existing but undocumented variable) default set to 1 for automatic abort on run-time error (error usually from lack of calculation time). Coordinate system "run-time-error" status bit left set after auto-abort.
6. Implemented 6 new DPRAM binary rotary buffers (8 total). This moves start of DPRAM data gathering buffer from $D200 to $D240; pointers from $D1FF to $D23F.
7. Implemented MACRO "Type 1" protocol.
8. Corrected suspension of position loop integrator operation on DAC saturation when Ix34=0
9. Corrected timer reset of I^2T function on $$$ reset.

10. Corrected rotary program buffer operation so that rollover of buffer does not count as "jump back" for purposes of "double jump back" blending stop.
11. Corrected intermittent background MACRO Type 1 data read problem.
12. Improved efficiency of background MACRO Type 0 data read/write.
13. Corrected intermittent problem in resolver absolute read.
14. Corrected intermittent problem in MACRO Yaskawa absolute encoder read.
15. Corrected power-on loop closing (Ix80=1) problem on MACRO systems.
16. Corrected **PRELUDE** operation when full program line is subroutine call.
17. Changed default I-variables on Ultralite PMAC2s so default setup is for MACRO system.
18. PMAC2 I9n6 default value set to 0 so PWM outputs are the default -- for protection of direct PWM amplifiers.
19. Corrected I^2T operation for PMAC(1) and non-commutated PMAC2 systems.

Update Summary: From V1.16C to V1.16D (Nov 1997)

1. Added geared resolver power-up position thru MACRO in Ix10 (=$73xxxx).
2. Fixed timing of VME DPRAM ASCII interrupt; held off until last interrupt acknowledged
3. Fixed binary rotary buffer transfer for high-speed host computers
4. Added run-time error codes in X:$0799: 1=insufficient calculation time; 2=program counter before start of program; 3=program counter past end of program; 4=unlinked conditional; 5=subroutine stack overflow; 6=label not found
5. Added parallel power-up position thru MACRO in Ix10 (=$74xxxx).
6. Added Yaskawa abs. enc. power-up phase position thru MACRO in Ix81 (=$72xxxx)
7. Added resolver power-up phase position thru MACRO in Ix81 (=$73xxxx)
8. Added parallel data power-up phase position thru MACRO in Ix81 (=$74xxxx).
9. Repeated clearing on software reset of general-purpose outputs on PMAC(1)
10. Fixed sign extension of Ix10 read 24-bit parallel feedback when negative
11. Added Sanyo absolute encoder power-up position in Ix10 (=$32xxxx).
12. Delayed testing for amp fault after enable thru MACRO (flags transferred every 2 cycles).
13. Computed post-trigger axis target positions on move-until trigger.
14. Permitted MACRO Station I-variables to be called "MIx", MACRO Station commands to be called "MCx" to better distinguish from PMAC variables.
15. Standardized Mini-PMAC(1) firmware
16. Added '$' to hex data returned from MACRO Station if I9=2 or 3, in reporting to host
17. Added more time to get power-up phase position from MACRO Station
18. Fixed direct microstepping operation on PMAC2.
19. Cleared MACRO output registers on reset to support new DSPGATE2B IC.
20. Fixed **SIZE** command response with buffer open
21. **DELETE GATHER** does not clear DPRAM gather pointers unless DPRAM gathering set
22. **GATHER** clears DPRAM gather index if DPRAM gathering set.
23. Made only Card 0 addressed at power-up/reset in daisychain mode (problem from V1.16A).
24. I51=1 does not automatically clear motor torque offsets each servo cycle.

Update Summary: From V1.16D to V1.16F (June 1999)

{Note: V1.16E was never released}

1. Fixed listing of **GOSUB{expression}** – had listed as **GOTO {expression}** .
2. Permitted more flexibility in detecting MACRO communications errors in I1001 servo cycles by use of new variables I1004 and I1005.
3. Made MACRO communications error checking more predictable by moving check from background to real-time interrupt
4. Permitted Ix10 absolute power-on read outside of full board reset, using new **$*** command.
5. Ix80 bit 2 set to 1 disables Ix10 absolute power-on read during full board reset.
6. Motor "home complete" status bit cleared at beginning of Ix10 absolute power-on read, set to 1 on completion of successful read.
7. On Mini-PMAC, permitted software-read jumpers to be changed and read properly on next reset, without cycling power.
8. On Mini-PMAC, always bypass routine to read (non-existent) control panel port, regardless of value of I2.
9. Permitted leadscrew compensation table to operate as function of net commanded position instead of actual position if 'D' character declared after source motor in **DEFINE COMP** command (e.g. **DEFINE COMP 100, #1D, #1, 500000**).
10. Fixed PMAC operation when accessing absolute position through MACRO nodes 8 and higher.
11. Refined interaction of overtravel limits with phasing search moves:
    – Limits are checked at power-up/reset before phasing search algorithm, preventing immediate phasing search if in limits.
    – Limit status is monitored even when amplifier is disabled, permitting motor to be manually moved off limits, then phased.
    – If phasing search move ends up in limit, motor is killed and phasing search error bit is set.
12. Implemented DPRAM ASCII interrupt with CTRL0 line on PMAC2-PC Ultralite (requires ECO #1282 to be seen by PC).
13. Corrected 2D leadscrew compensation table so could be non-square dimension in counts and number of entries.
14. Implemented encoder conversion table entry for high-resolution analog encoder. Format $F0. Two-line entry. First line is $F0xxxx, where xxxx is the address of the encoder registers (e.g.$C020 on ACC-51). Second line is $00yyyy, where yyyy is the first (lower) address of the A/D input pair (e.g. $C022 on ACC-51)
15. Modified $50 encoder conversion table entry for integrated A/D conversions to mask out lower 8 bits before integrating. Important for proper interface of CAN-drive system.
16. Motion program can now progress through non-motion and **DWELL** commands when % value is 0. Permits viewing of calculated move end position even when **%0** prevents interpolation from starting.
17. Added **PAUSE PLC n** and **RESUME PLC n** commands (on-line, motion program, and PLC program) to allow suspension of PLC operation without having to restart at the top.
18. Enhanced **LIST PLC n** command to permit listing from point of pending execution.
19. Added I63 variable; when set to 1, PMAC echoes <CTRL-X> character back to host, so host knows when communications buffers are clear.
20. Ix26 home offset parameter now also used on Ix10 absolute position read, subtracted from sensor position to get motor position.
21. Permitted Type 1 MACRO auxiliary command to multiple MACRO Stations with **MS15, MIx={data}** , for "anynode" MI-variables on the MACRO Stations.
22. Corrected location of buffer pointers in memory map (wrong in V1.16C and D)
23. Corrected operation of **PRELUDE** function for code and program numbers less than 8
24. Values of I53 and I62 are now stored in a **SAVE** command.

25. If vector distance of feedrate axes in a feedrate-specified move is 0, the programmed feedrate is used to control the speed of the non-feedrate axes in the move. The axis with the longest distance is moved at the programmed feedrate; other axes move with the same move time.
26. When in feed hold mode and no program running (also no motor jogging), a Run or Step command now starts program in addition to clearing from feed hold mode.
27. Implemented Ix98 maximum feedrate value; F commands in program compared to this value; if greater than Ix98, Ix98 is used instead.
28. Fixed operation of synchronous M-variable assignments when move times or acceleration times smaller than I13 time.
29. Added I64 variable; when set to 1, PMAC leads unsolicited responses (from program **SEND** or **CMD** statements) with <CTRL-B>. To support this feature from compiled PLCs, PCOMM32 version 2.21 or newer (March 1999 or later) is required.
30. When I58 is set to 1 to enable DPRAM ASCII communications, PMAC no longer forces values for I3, I4, and I6. DPRAM formatting unchanged, except that if I6=2 when I58=1, PMAC no longer reports errors into DPRAM for illegal internal commands.
31. Run-time error always triggers abort, regardless of setting of Ix97.
32. Implemented variable full-circle threshold value with I90 to permit user to define minimum arc length.
33. Cutter compensation refinements:
    - If cutter compensation is active and PMAC cannot find the next move in the compensation plane, PMAC no longer removes the compensation at the end of the move; instead it ends the move at the proper point to start an outside corner.
    – If cutter compensation direction is changed with compensation active, offset to new direction occurs at move boundary, not over the course of the next move (unless there is no intersection of compensated paths, in which case the change still occurs over the next move).
    – 180° reversal with arc(s) now treated as inside corner, not outside corner
    – **DELAY** move timing while in compensation corrected.
34. **PRELUDE** operation refined so that **<CR>** always starts new subroutine call.
35. **READ** operation now stops on repeated character.

Update Summary: From V1.16F to V1.16G (Sept 1999)

1. Fixed operation of **NORMAL** commands for non-default settings
2. Fixed operation of **DEFINE GATHER** when gathering set up for dual-ported RAM (I45=2 or 3) so that it reserves entire open space in DPRAM.
3. (Option 6L only) Fixed intermittent problem when re-starting operation in lookahead after feedhold.
4. (Option 6L only) Permitted jogging away from feedhold position in segmented **LINEAR** or **CIRCLE** mode move, or **SPLINE** move.
5. **MSDATE** command to MACRO Station now returns 4-digit year value.
6. Added **EAVERSION** command to combine aspects of **VERSION** and **TYPE** command.

Update Summary: From V1.16G to V1.16H (Sept 2000)

1. Fixed problem when issuing a **CMD"PMATCH"** from within a motion program.
2. Fixed floating-point underflow problem that could produce very large values.
3. Fixed operation of homing-search move if previous homing-search move was interrupted by a **K** (kill) command.
4. Improved accuracy of high-resolution encoder interpolation calculations in conversion table.
5. Responses to **CMD "{on-line command}"** are no longer terminated with an <ACK> even when I3=2.
6. Fixed scaling of parallel absolute position received through MACRO ring in response to **$*** command.
7. Fixed rare intermittent problem in response to **MACROSLVREAD** command.
8. Fixed intermittent problem in action of the **CLOSE** command.
9. Fixed operation of resolver-to-digital converter read through MACRO ring with **$** command.
10. Fixed small issues with cutter radius compensation.
11. Fixed operation of leadscrew compensation tables in center one-eight-millionth section.
12. On PMAC2 boards jumpered for "external clock" with E1, gate array ICs are automatically set up to input clocks.
13. If I^2T feature is disabled by setting Ix57 to 0, I^2T fault bit is automatically cleared.
14. **J!** command always forces command position to nearest full count, regardless of following error.
15. Fixed operation of single-line **IF** and **WHILE** statements in rotary buffer when there are no subsequent lines in the buffer.
16. PMAC2 Ultralite no longer needs to enable motor with "**O0**" command before issuing "**$**" command to phase motor.
17. Auto-detects flash IC type (AMD or SST) so common firmware can save to either type of IC.
18. Implemented alternate rollover mode for rotary axes A, B, and C. If Ix27 is set to a negative value, the sign of the axis command destination value represents the direction to travel to the destination.
19. Implemented **SETPHASE** command (on-line, motion, PLC), which forces Ix75 value into phase position register.
20. Implemented **OPEN BINARY ROTARY** command to permit simultaneous entry of binary buffered commands and ASCII on-line commands.

Update Summary: From V1.16H to V1.17 (Oct 2001, FLEX CPU only)

1. Added support for Option 5xF "Flex" CPUs with DSP56300-family CPUs.
2. Variable I0 now specifies serial addressing card number (@n) for PMAC(1) boards with Flex CPU (as well as for all PMAC2 boards).
3. New variable I46 sets CPU operational frequency for boards with Flex CPU. If I46=0, jumpers are used for this
4. Variable I54 now specifies serial baud rate for PMAC(1) boards with Flex CPU and I46 > 0 (as well as for all PMAC2 boards).
5. Fixed response to **LIST COMP DEF** command.
6. Firmware reference checksum value now included as part of firmware itself. It is no longer required to re-initialize the card to compute the reference checksum.
7. New **CHECKSUM** on-line command causes PMAC to report the firmware reference checksum.

Update Summary: From V1.17 to V1.17A (Jan 2002, FLEX CPU only)

1. New variable I66 permits disabling of autocopy of servo-channel ADC registers to RAM each phase cycle. This autocopying is no longer necessary for robust background reads of these registers, so disabling saves time and permits proper interface of ACC-51P interpolator ADCs.
2. Permitted PMAC to accept comments after M-variable definition statements without returning an error.

Update Summary: From V1.17A to V1.17B (Sep 2002, FLEX CPU only)

1. Corrected interrupt-blocking for Flex CPUs, fixing problem in transition between LINEAR and SPLINE modes
2. Refined timing on JTHW multiplexer port for use of ACC-34 boards with fastest Flex CPUs.
3. Added purge of host port during power-up/reset to ensure serial port comes up as the default response port.
4. On Ultralite boards with Flex CPUs, force any additional (unused) MACRO CPUs to input clock signals to prevent possible contention if these are installed.
5. Improved power-on dual-ported RAM detection test for new interface buses.

Update Summary: From V1.17B to V1.17C (Sept. 2005, FLEX CPU only)

1. Added support for new controller configurations Mini-PMAC-PCI and Geo PMAC integrated amplifier.
2. (Lookahead firmware only) Corrected problem in jogging motors during stop in lookahead where motors could be de-activated.
3. Corrected minimum move size for rotary axis with Ix27 < 0.
4. Corrected position following function with Ix08 <0.
5. Corrected high-resolution interpolation (conversion method $F) with PMAC2-style ASIC.
6. Corrected handling of comment after M-variable definition on USB port.
7. Corrected auto-detection of SST-type flash IC.
8. Corrected data corruption problems on interrupt.
9. Corrected power-on DPRAM detection problem.
10. Made timing for I/O ports on PMAC(1) more robust with 160 MHz CPU (Opt 5EF).
11. Corrected operation of program homing search move after move-until-trigger that did not find trigger.
12. Embedded checksum into firmware file, so **$$$***** command is not required after firmware download to establish proper checksum match.
13. Added "do-nothing" variable I65 to give user easy way to confirm if application configuration has been downloaded and saved, and/or to serialize controllers.
14. Assigned "hardware 1/T enable" control bit to new I-variable I9n9 for Channel n (PMAC2 only).
15. Assigned "third-channel demux enable" control bit to new second bit (bit 1) of existing I-variable I9n5 for Channel n (PMAC2 only).
16. Assigned ADC data table setup in X:$0708 – X:$070F to new I-variables I70 – I77, respectively.
17. Added support for byte-wide parallel data reads to support ACC-14P boards in encoder conversion table methods $2 and $3.
18. Modified action of "MaxChange" filter in conversion table methods $3 and $7. If MaxChange is exceeded, result is now changed by "LastChange", not "MaxChange", if previous cycle's result was good.
19. Added I-variables I1010 – I1012 to support resolver excitation (Geo PMAC only).
20. Added I-variable I1013 to support motor temperature check (Geo PMAC only).
21. Added I-variables I1015 – I1019 to support SSI absolute encoder interface (Geo PMAC only).
22. Added five-line tracking-filter variant of exponential-filter conversion method ($D) enabled by setting bit 19 of first setup line to 1, permitting low-pass filter without steady-state error – suitable for processing feedback data.
23. Added resolver conversion method ($E) to encoder conversion table (Geo PMAC only).
24. Added three-line interpolated sinusoidal encoder conversion method ($F) with sine/cosine bias word (Geo PMAC only – this method is a two line entry without a bias word on other PMACs).

25. Added five-line diagnostic method entry for highly interpolated sinusoidal encoders ($F) enabled by setting bit 19 of first setup line to 1, to calculate sum-of-squares magnitude or sine/cosine bias word (Geo PMAC only).
26. Added amplifier status/fault code for seven-segment display (Geo PMAC only), including **CLEARFAULT** command to reset display.
27. Added support for hardware position capture with highly interpolated sinusoidal encoders and capture of estimated sub-count position data in triggered moves with bits 18 and 19 of Ix03.
28. Removed obsolete **LIST LDS**, **<CTRL-E>** and **<CTRL-W>** on-line commands and I47 variable.
29. Corrected action of true deadband (Ix64=-16) with pulse-and-direction output to eliminate potential for dithering.